PaintBoard

Prototyping Interactive Character Behaviours by Digitally Painting Storyboards

by

Daniel J. Rea

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

In partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

Department of Computer Science

University of Manitoba

Winnipeg

Advisor

Dr. James E. Young

Author

Daniel J. Rea

PaintBoard: Prototyping Interactive Character Behaviours by Digitally Painting Storyboards

Abstract

The creation of interactive worlds, such as those in video games, often include a set of computer controlled characters that must intelligently act and react in response to dynamic input from the user. These *interactive behaviours* usually require authors to programmatically define each behavior, reaction, and interaction the character should take in response to user input across a range of scenarios, a process that can take significant time. While this method can successfully create robust characters, the large development overhead is not conducive to the exploration, prototyping, and testing of new character ideas.

We designed and developed PaintBoard, a system that enables users to rapidly prototype interactive character movement by digitally painting a storyboard. PaintBoard promotes prototyping by facilitating quick, visual authoring, and by enabling immediate testing by allowing the user to interact in real-time with a behaviour generated from a painted storyboard. To generate the interactive behavior, we developed a novel algorithm that analyzes a painted storyboard and uses machine-learning to generalize the painted examples to new situations. Our algorithm uses

this generalized behaviour to control the computer character during real-time interaction with the user character.

To help ground our design decisions in how real designers approach the problem of creating interactive behaviours, we conducted two preliminary exploratory studies with industry professionals and programmers. We further conducted a proof-of-concept workshop with our prototype to investigate how real developers may use PaintBoard; this illustrated the initial success of our painting-storyboards authoring metaphor. Finally, we performed an initial evaluation of the behaviour generation algorithm which informed us of directions for future work to improve PaintBoard's performance.

Contributions of this work include the design and evaluation of both a new interaction metaphor—digitally painting storyboards for interactive behaviour authoring—and a novel behaviour generation algorithm (for generating real-time interactive behaviours from storyboards). Our results demonstrate that the PaintBoard approach can be useful to developers as an exploratory prototyping tool due to its fast and understandable painting metaphor, and that PaintBoard itself can quickly (in real-time) generate an interactive behavior.

Acknowledgements

Over the past two years, I have been fortunate enough in my professional and social life to meet many wonderful people who have helped me succeed during my master's degree, and thanks are in order.

I would first and foremost like to thank Jim Young, my supervisor, for all the advice, the pep talks, the brainstorming, and yes, even the 26739 edits that helped me go from thinking I know how to write, to realizing I still have a long way to go (and not just in writing!). Thanks especially for all the opportunities you've given me, even when you had to drag me kicking and screaming to some of them. The echoes of these 2 years will undoubtedly be with me for a long time to come.

I would also like to thank Takeo Igarashi, who was my co-supervisor in all but name. Your research is inspirational, and your advice throughout the project truly set me up to succeed.

Thank you to my committee, Andrea Bunt, and Jonah Corne. Your experience and perspectives really helped fill out and polish my research, and your guidance was always helpful.

Of course, I would never have gotten through this without my family. Thanks to my mom for listening to all the rants and times I was stumped, even if you didn't understand. Thanks to my dad for always quietly supporting me; sometimes the talk and a coffee helped me get through the next week. Thanks to my sister, who listened, understood, and supported me; who caused trouble and shenanigans with me; who drank bubble tea, and ate sushi; who took me out made me laugh when nothing seemed to be going right: you're the best.

Thanks to all my friends for being patient. I cancelled a lot of plans throughout this degree, and you always understood and stayed with me through it all. There's too many of you to thank individually, but I'm thinking of you.

To all you crazy people in that sharehouse: thank you so much. I don't know where I'd be or who I would be now if I hadn't met you. You made my Tokyo internship unbelievably fun, warm, and full of adventure. Those months will be in my heart, and I hope we can meet again somewhere in the world, someday soon!

To all the researchers I've met throughout these years (and even during my undergrad), you don't know it, but you're the reason I'm still here, and the reason I started this journey in the first place. I don't know if we'll see each other again, but you're an inspiration to me.

Thank you, all of my labmates, for the movie nights, the LAN parties, the coffee trips and bubble tea. Thank you for the board games the fun talks, the yelling, the wasted afternoons, and the help. Thank you for being awesome. You've made the 2 hours of travel every day worth it.

Thanks to my cats. I'm sorry I never wrote any papers with you.

And to everyone else who I helped me somewhere along the way; keep on rockin'.

Dedicated to these two years.

“It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.”

— J.R.R. Tolkien, The Lord of the Rings

Table of Contents

List of Tables

List of Figures

Publications

Some ideas and figures in this thesis have appeared previously in the following publications by the author.

Daniel J. Rea, Takeo Igarashi, James E. Young. Behavior Primitives for End-User NPC Behavior Creation. *In adjunct proceedings of the international conference on Human-Agent Interaction (HAI '13)*, 2013. (Best Poster Runner Up)

Daniel J. Rea, Takeo Igarashi, James E. Young. PaintBoard - Prototyping Interactive Character Behaviors by Digitally Painting Storyboards. *In proceedings of the international conference on Human-Agent Interaction (HAI '14)*, 2014. (Best Paper Award)

1. Introduction

Interactive media has become a part of our everyday lives, and is being used in applications such as video games, art, and training simulations. Computer controlled characters—those that have an artificial intelligence and move around and interact with the player and the environment—are an important element of many of these interactive worlds. The creation of these characters is a complex task that, at the professional level, can require a broad range of specialized and collaborating experts, including artists for authoring 3D models and animations, writers and voice actors for dialogue, and programmers to implement artificial intelligence and system logic. When the computer-controlled characters are highly interactive, the artificial intelligence component becomes particularly challenging as the characters must assess and interact appropriately in real-time to dynamic input from users and their environment. The creation of *interactive behaviours* can demand significant amounts of time even from expert programmers. For example, in a role playing game, a designer may want a computer-controlled thief character to "sneak": avoid the user-controlled character when they are nearby while simultaneously approaching a treasure box, all without being seen. Such behaviours usually require the designer to logically (programmatically) define the details of the computer character's actions for multiple situations

based on the user character's past and potential activity. This overhead slows development; however, the ability to quickly create and explore multiple ideas is important for creative tasks [34].

To aid in the prototyping of digital content for interactive systems, researchers have aimed to reduce the amount of expertise and time required to make them. This approach includes enabling people to create 3D models simply by sketching in 2D [21], author advanced animations through simple mouse or touch manipulations [22,23], or to create complex interactive stories through point-and-click visual logic programming [28]. Simplifying the creative process provides experts with prototyping tools for quickly testing, visualizing, and sharing their ideas [34]. These methods also have the added advantage of improving the accessibility of content creation and prototyping to potential authors who may not have the required expertise. For example, they can help non-technical artists program logic, or programmers create 3D models. We extend this body of work by introducing PaintBoard: a system that enables authors to rapidly prototype interactive computer controlled character behaviours without programming, simply by digitally painting a storyboard of a behaviour.

Low-fidelity iterative design techniques such as paper sketching and storyboarding are low-cost, fast, easy-to-use tools that support creativity and exploration [5,27,34] by enabling rapid iteration of ideas [5], and by providing immediate visual feedback of those ideas [5,27,34]. These approaches also inherently support storytelling, communication, and discussion of design ideas with others [18]. Because of their utility, such low-fidelity techniques are found in standard toolkits across a wide range of fields that includes human-computer interaction [21,27,41], film [17], animation [22,42], and software development [9,27]. Our PaintBoard system and approach

leverages these exploration techniques to enable authors to create interactive behaviours by roughly painting ideas on digital storyboards, a process similar to sketching, and by immediately generating results that people can interact with, test, refine, iterate over, and show to others.

In this work, we present our PaintBoard prototype that provides a simple storyboarding interface for painting interactive behaviors (in our case, discrete character movement on a 2D grid), as seen in Figure 1.1 below. To better understand possible roadblocks encountered when creating interactive behaviours, we conducted interviews with industry professionals which informed us that exploring and prototyping behaviours, as well as communicating desired behaviours to colleagues were important bottlenecks in the creation process. In addition, to guide both the PaintBoard interface and algorithm design, we conducted a behavior-programming workshop with a group of experienced programmers and analyzed common approaches to designing and creating interactive behaviours. From this analysis, we developed a feature set for representing interactive behaviors, and a machine-learning algorithm that uses these features to generate real-time interactive behaviors from the user-authored storyboards. As an initial investigation of the approachability of the painting-storyboards method, we conducted a proof-of-concept hands-on PaintBoard workshop that highlighted the potential of our techniques. Finally, to better understand the strengths and weaknesses of our algorithmic approach, we conducted an initial evaluation where we compared our algorithm's performance when used with different underlying configurations and parameters.

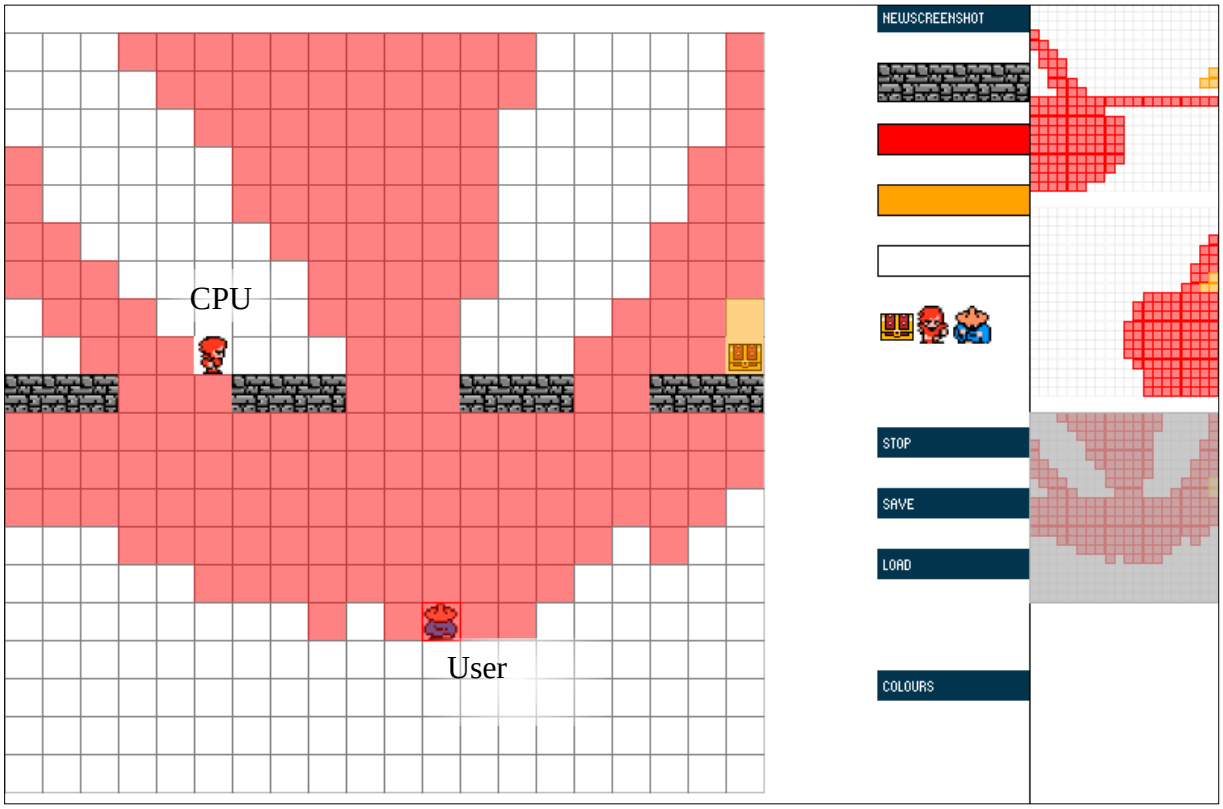

Figure 1.1: A painted storyboard showing that a computer-controlled interactive character (CPU) should approach yellow squares (the treasure) while staying out of the red squares (user controlled character's sight). PaintBoard extrapolates and generates the interactive behavior of "sneaking to the treasure when the user's character is not looking."

1.1. Methodology

Our approach to the creation of PaintBoard follows a user-centered design approach [3] where we first ground our work in the needs of potential users, use this information to build a prototype, and have potential users evaluate that prototype. To help guide our interface and algorithm design choices we conducted two initial exploratory studies: interviews with industry professionals to discover possible difficulties during the design of interactive behaviours, and a behaviour-programming workshop with experienced programmers to investigate how some programmers are

already creating interactive behaviours. Our findings aided our prototype design, for which we developed both a novel interface technique for authoring interactive behaviours and an algorithm that can generate an interactive behaviour from a painted storyboard and can test it in real-time. We evaluated our prototype in two phases: we conducted a hands-on PaintBoard workshop to evaluate our storyboard-painting authoring technique, and we performed an evaluation of our algorithm by comparing the behaviours generated by different configurations of our system.

1.1.1. Exploratory Investigations

To investigate problems that may occur when creating interactive behaviours and the workflows used to create them, we conducted interviews with professionals from the video game industry. Interview questions focused on general approaches to implementation (e.g. programming and design) and workflows (e.g. design process of interactive behaviours, communication between designers and developers) used when creating interactive behaviours. We audio recorded the interviews, and manually transcribed them for analysis; our analysis method was inspired by open coding techniques [2] where we iteratively grouped data by tagging themes in participants' interview answers and let salient themes emerge from the data. Section 3.1 describes the interview design, analysis, and results of the interviews.

As programmers are already able to create compelling interactive behaviours in code (for example, the video game series *Assassin's Creed*[1] is noted for its realistic crowds of computer controlled characters [29]), in addition to our interviews we decided to explore programmatic approaches in

[1] http://assassinscreed.com/

order to help inform our PaintBoard prototype's interface design and algorithm design. For example, investigating how programmers conceptually frame behaviours could improve our interface design; investigating how programmers defined their behaviour in code could inform how our algorithm analyzes an in-game scene. To this end, we conducted a programming workshop where we recruited experienced programmers to create interactive behaviours in computer code, which we analyzed to identify common implementation techniques.

We analyzed the implementations themselves, with exploration and grouping techniques inspired by open coding [2] in order to better understand concepts in interactive behaviours that could be included in our authoring interface. Additionally, we looked for common approaches for computationally describing behaviours in order to inspire our behaviour generation algorithm. We analyzed the types of behaviours we received by iteratively grouping similar behaviours together to better understand the variety of possible behaviours, as well as to build a set of target behaviours we could test during development. The details of the study, analysis, and results of the programming workshop can be found in Section 3.2.

1.1.2. PaintBoard Prototype Design and Development

We used the knowledge from our exploratory investigations to ground the development of our PaintBoard prototype. Our prototype had the following design goal: PaintBoard should enable the rapid prototyping of interactive behaviours. We decided to allow users to digitally paint storyboards of the behaviour to leverage the benefits of low-fidelity techniques [5,27,34]; painting allows content to be created in quick strokes. Storyboarding was chosen to allow complex behaviours—such as sneaking around a castle while staying out of sight and trying to find

treasure—to be expressed piece-wise as frames in a storyboard. PaintBoard's behaviour generation algorithm was designed to analyze a painted storyboard as training data by calculating a set of training features, and output an interactive behaviour.

The development of PaintBoard presented interface problems—how the painting metaphor can be leveraged to enable users to define interactive behaviours—and algorithmic problems—how our algorithm analyzes the painted storyboard, and how it generates the behaviour from this input. Details of these components are explained in Section 4.

1.1.3. Evaluation of the Painting-Storyboards Technique

We conducted an exploratory workshop with professional and hobbyist game developers to investigate their ability to use PaintBoard, as well as to elicit free-form feedback on our overall approach, interface, and interaction design. Specifically, we explored if our users think PaintBoard's approach to painting and storyboarding is a useful way to prototype interactive behaviours, collected sample behaviours that our users reported were easy or hard to create with PaintBoard, and inquired as to how users think PaintBoard may fit into their workflows. The goal of the study was to be a proof-of-concept with a sample user set, and to provide initial insight into our chosen approach.

In the workshop, our participants used PaintBoard to create interactive behaviours of their choice. The behaviour storyboards, field notes by the on-site researcher, and post-study questionnaire responses from each participant were analyzed to explore their experience with our system. The qualitative analysis method was an open-coding-inspired [2] technique where we iteratively tagged

data to let themes emerge. Our analysis helped us understand some of the possibilities of using painting storyboards as an interactive behaviour creation tool, where its limitations lie, and how we may improve it in future work. Full details are given in Section 5.

1.1.4. Evaluating the Accuracy of Generated Behaviours

We compared the behaviour output of different underlying machine learning classifiers and training feature sets to understand how our algorithmic choices impacted the accuracy of behaviours generated by PaintBoard. For this comparison, we required a dataset of painted storyboards that allowed us to train and test our algorithm in an ecologically valid way. It was also necessary to devise a performance metric for measuring the accuracy of a predicted behaviour. With the dataset and accuracy measure, we performed our comparisons, and our analysis revealed how we may improve the PaintBoard's behaviour generation in the future.

From our initial investigations (Section 1.1.1), we knew that the interpretation of a single behaviour type, such as "sneak to the user," was often different between authors. Thus, we decided to create a dataset where each behaviour had sufficient storyboard examples from one author to both train our algorithm, and to test the results. Recruited participants each authored many storyboards of a behaviour. We chose the test and training sets from these storyboards using a variant of k-fold cross validation [26]. For each algorithm, we analyzed the performance results across authors and behaviours to get a measure of the algorithm's accuracy.

In a separate analysis, we varied the feature set used to analyze the storyboard data. This was to better understand the impact of each training feature, and if modifying feature sets was a useful

direction to improve PaintBoard's behaviour generation. We designed a simple greedy feature selection, which helped us arrive at a smaller feature set with similar performance to our full set.

Measuring each algorithm's performance was a non-trivial challenge: we needed a metric that enabled us to compare generated output (the behaviour) to a ground truth. With painted storyboards, no such trivial comparison point exists; for example, there is no straight-forward distance function that can be applied to a behaviour (PaintBoard's output) and a painted storyboard. Thus, part of our evaluation involved the development of such a metric that could be used to calculate the accuracy of an algorithm variant. Full details of the analysis are provided in Section 6.

1.2. Contributions

Contributions of our research include:

1. A novel interaction method—digitally painting storyboards—that enables the description and real-time testing of interactive behaviours for computer controlled characters.

2. An original interface that enables 1.

3. A behaviour generation algorithm that can quickly generate a real-time interactive behaviour from a painted storyboard.

4. A workshop with developers and interviews with professionals in the game industry that grounded our design and development of 1, 2, and 3, and provide a baseline understanding of some approaches of creating interactive behaviours.

5. An evaluation of PaintBoard's overall approach of painting storyboards

6. An evaluation of PaintBoard's behaviour generation algorithm.

Our research explored how to apply digital painting and storyboarding to aid the rapid design and prototyping of interactive behaviours for computer controlled characters. Through this research, we created a novel interface prototype that serves as a proof of concept for how users can paint storyboards of interactive behaviours, and a novel algorithm to analyze users' storyboards and generate interactive behaviours in real-time. We evaluated our approach and interface with a workshop study, and compared how several different machine learning techniques generalize user storyboard data to create interactive behaviours. We believe our results will be of interest to designers of interactive systems such as training programs, public art installations, robots, or video games.

2. Related Work

The field of Human-Computer interaction (HCI) has many focuses, including understanding how people use technology and designing new software and hardware interfaces [10]. One particular tradition is the development of novel interaction techniques that decrease the complexity of tasks that require computers. Decreasing complexity allows task-experts to be faster and more efficient, and often enables non-expert users to perform similar tasks [10]. One area of complex tasks that use software is the creation of digital content, and of particular interest to this work is game-related content such as 3D character and environment models, and programmed game logic.

Creating complex content such as 3D worlds, virtual characters, or creating stories for the characters in these worlds, often requires skilled designers with a variety of skills. For example, writers could write the story and dialogue for characters whose 3D appearance and movement is created by animators, and a user could interact with these characters through logic written by a programmer. Researchers have simplified the creation of such digital content by reducing the amount of experience and skill required to create them, as traditional mediums such as powerful 3D modelling applications or writing computer code can take hundreds of hours of training to reach professional proficiency. For example, current research aids authors in exploring storylines

[30], enables 3D models and animations to be created in minutes instead of hours without complex tools [21,22,23], and the game worlds and the logic that runs them can be specified without knowing a programming language [28]. By reducing the required skill, such research enables faster authoring as creators can focus more on content instead of technique. This faster authoring in turn increases the pace of iteration and the ability to explore different ideas [5,34] by freeing up the authors' time and resources. An additional benefit of these approaches is the lower barrier of entry: it is easier for non-experts to author content. We employ this overall approach in PaintBoard to a problem that has not yet been addressed: lowering the complexity of authoring interactive behaviours for computer characters in interactive systems.

One way researchers have been able to simplify the design of complex content is to perform low-fidelity prototyping. This technique enables the rapid creation of content at the expense of a detailed and precise final result [5]. Such a trade-off is useful for the initial exploration of ideas, where refined details are not as important [5,34]; these techniques have been successfully leveraged by many researchers in various domains [9,17,18,21,22,27] and are accessible as design and prototyping tools for both professional and amateur designers. PaintBoard extends this body of work to a new domain by applying low-fidelity prototyping techniques to the design of interactive behaviours for computer-controlled characters.

Storyboarding and sketching are low-fidelity prototyping techniques that have been used to successfully simplify the design of other forms of digital content. Sketching, for example, has been used to create 3D character models by sketching simpler 2D representations [21], and sketched user interfaces for desktop computer software can be turned into working prototypes [27]. Traditional paper-based storyboarding is useful for planning out long, complex sequences of

actions and has been used to aid interface design [18], video editing [17], animation [11], and software systems [17]. Painting, similar to sketching, has been used as a real-time control method for non-interactive computer-animated characters [37]. Another key element of these works is that they keep their interfaces relatively simple, for example, one easy-to-use 3D modelling tool, even with extra features added, has far less options than modern professional tools such as Maya[2] [20]. Our proposed method unifies these approaches and applies them for the first time to the creation of interactive motion behaviours.

Existing research that simplifies the creation of interactive characters and systems has aimed to reduce the programming requirements typical to such tasks. One approach for simplifying the creation of interactive behaviours focuses on programming-assistance tools, such as tools in software development environments that automatically generate computer code from programmer-specified relationships between game elements [19]. A related approach is to create programming languages specifically for programming behaviours [25,38]. These approaches succeed in reducing the difficulty of programming, but still require logical, step-wise specification of behaviours. Visual programming, where designers use drag-and-drop, visual representations of game objects combined with programming-like elements, has been used to simplify the creation of interactive worlds and the ways users can interact with the world to progress through a game's story [28] (their focus was not on interactive behaviour movement). Another work allows authors to explore and create complex and varying narratives for interactive stories after specifying the details about the characters and the world they act in, such as characters' psychology, goals, and how the characters can change their world [30]. Other research has found that detail-oriented thinking like

[2] http://www.autodesk.com/products/autodesk-maya/overview

many programming techniques may slow down prototyping and exploration [36]. Thus, in our work, we aim to avoid the requirement for users to explain their behaviour logically and enable them to create it in a less detail-oriented and story-like way.

Research has discovered that *in-situ authoring*, an interface design technique where content authors can create, preview, and edit their content, all in the same environment, helps speed up content creation [3]. The speed up is thought to be because authors do not have to mentally translate from their creation medium to the final result. For example, it might be difficult for a programmer to imagine how an interactive behaviour may feel to a user in the game if the programmer makes a change to textual programming code. Researchers have successfully used in-situ design to aid visual interactive behaviour authoring [33]; their approach, however, requires users to create behaviours by entering commands in a valid sequences (a state machine), similar to programming. With PaintBoard, we aim to use an in-situ authoring approach by having authors create, edit, and test the behaviour in the environment that the end-user of the interactive system will view it in. Additionally, in contrast to previous work [33], we designed PaintBoard to avoid strict linear behaviour authoring techniques: research has suggested that linear thinking holds inhibits creativity [36]. PaintBoard enables this non-linear authoring by allowing users to paint, create environments, edit behaviour storyboards and interact with their behaviour in any order at any time.

Programming by demonstration, where a designer can author a behaviour by simply providing a performance demonstration, is an inherently in-situ technique that does not require programming-like thinking. Although this approach is well established, most of this work has been for the creation of static behaviours without an interactive element. For example, researchers have enabled

users to create behaviours for robots when the single goal and action to be performed are well defined, for example, picking up certain types of objects out of a group [35]; in animation, authors have used interactive, demonstration-based techniques to animate characters interacting with other characters in order to produce static videos [11,22]. Interactive work has focused on, for example, learning reactive body language by recording movement with expensive motion-capture equipment [12], or learning well-defined sequences of actions that fit into a state-machine model [15,33]. PaintBoard extends this work by targeting the design of character movement that responds to a dynamic user character in the context of an environment (buildings, objects), without expensive equipment such as motion capture devices.

Further, a common problem encountered with programming by demonstration is that the techniques can often require large numbers of repetitive demonstrations (e.g. [13]). Often it is important to use real world data of many types of demonstrations (e.g. [24]), which can take time to acquire. Other work still uses programming after the demonstration, which may require programming expertise to use effectively (e.g. [40]). To enable its use as a low cost, rapid-prototyping tool, we designed PaintBoard to work with as little as one example, and avoided including computer programming in the authoring process.

Perhaps closest to our work is the Puppet Master programing-by-demonstration project [41], which enables authors to rapidly prototype interactive animation or robotic motion behaviors similar to the ones targeted by PaintBoard. While Puppet Master emphasizes interactive movement “style” (texture) of two interacting characters, PaintBoard builds upon their results to cover multi-part behaviors (e.g., hide when seen, get some treasure when guards are not looking), and enable characters to interact with the environment (e.g., walls, important objects such as treasure chests).

Further, Puppet Master's evaluations indicated that users had difficulty envisioning what behavior would result from their performance demonstration due to the mental load from Puppet Master requiring the user to successfully author the whole behavior in real-time in one attempt. To avoid such issues, we purposefully avoid direct performance demonstration and propose a novel approach by enabling more complex interactive behaviour authoring (with an environment, objects in the scene, etc.), that allows the behaviour to be created at the author's pace in visual frame-by-frame storyboards.

In summary, PaintBoard continues the research theme of reducing the expertise required for creating digital content (in our case, prototyping interactive behaviours). By combining the low-fidelity prototyping techniques painting and storyboarding with in-situ authoring, PaintBoard aims to speed up the exploration process that is important in the early stages of creative work. PaintBoard is also designed with a goal to enable non-linear creation styles by allowing behaviour editing and testing at any time. We extend previous work in programming demonstration by designing PaintBoard to produce a prototype interactive behaviour with as little as one behaviour example, and without requiring a real-time demonstration. Finally, PaintBoard builds upon previous interactive behaviour work by enabling the creation of behaviours that interact with an environment (walls, objects in the environment, etc.).

3. Exploratory Investigations

Our interface and algorithm design was guided by two exploratory studies. First, we performed semi-structured interviews with video game designers and developers to uncover common problems they faced during the creation of interactive behaviours, as well as workflows used to overcome them. Qualitative analysis of these interviews resulted in high-level goals for PaintBoard's approach to authoring interactive behaviours: enable rapid prototyping and better communication.

To inform our interface and algorithm implementations, we conducted a programming workshop through which we explored the range of interactive behaviors people may author, and analyzed implementations (computer code) to extract strategies and techniques used to implement them. This led to the identification of behaviour design patterns, such as goal-oriented behaviours (for example, stay close to the user character), which were included in our interface design. Additionally, we generalized the set of calculations commonly used by participants to define behaviours. This helped inform our algorithm design, which used some of the calculations to analyze in-game situations.

3.1. Interviews with Industry Game Designers

To investigate possible workflows used and problems encountered by professionals when creating interactive behaviours, we conducted interviews with four professionals (developers and designers) from the video game industry. We focused questions on general approaches to implementation (e.g. programming and design) and workflows (e.g. design process of interactive behaviours, communication between designers and developers) used when creating interactive behaviours. This information helped ground our design in the problems of real interactive system designers.

To accomplish this exploration, we conducted semi-structured interviews where we prepared a list of rough goals for the interview and questions to ask the participants; during the interview, when any of the answers provided interesting or new data, the interviewer explored further investigated the new aspect by improvising new questions, deviating from their interview plan [39]. The interviews were one-hour long, and our participants were four professional game designers and developers. Questions we asked include "How are certain interactive behaviours difficult to create, specifically because of interactions with the player?" and "How do you create these interactive behaviours?" The interviews were conducted by phone, and were audio recorded and then manually transcribed for analysis to explore some problems and workflows that we could target with PaintBoard; our qualitative analysis method was inspired by open coding techniques [2] where we iteratively grouped data by tagging themes in participants' interview answers and let broad salient themes emerge from the data. Please see Appendix A for materials used in the study, as well as proof of approval by the Joint-Faculty Research Ethics Board.

In the interviews, participants reported spending significant time planning how a behavior would act and react to the user character. Participants said they did this planning because planning beforehand can save time because actually implementing behaviours with programming takes time:

> *"I spend a lot of time thinking about what kind of rules to use and what kind of system to use them in to get the results that I want." – P1*

In addition, participants heavily relied on experimentation and iterative prototyping; they would write programs, interact with the results, tweak parameters or write different solutions, and iterate:

> *"That whole process that I like which is just seeing what it's like and then adjusting. It doesn't work that well because you think 'Ugh. Do I really want to make this adjustment? Because then I'm going to have to change 8 different [behaviour] states and track down bugs.'" – P2*

Further, participants reported having difficulty communicating their behaviour to others, and understanding how an interactive behavior described by another should look. We saw this reported by both technical developers as well as artistic designers.

> *"Designers will oftentimes knowingly work around bugs in the system…they're very reluctant to bring things up…It could be the type of the thing you could fix, as a programmer, in 15 minutes" – P3*

This developer thought a feature was implemented correctly, but the designer authoring the behaviour did not collaborate well when things went wrong. The developer desired the designers to better communicate what they wanted. A designer we interviewed did communicate often with developers, but found it difficult due to behaviours being hard to describe:

"Basically I describe the components...in as much detail as seems useful. Then [the developers] would interpret it and implement it, then I'd play it and give additional feedback from there. Just verbal feedback...We do this back and forth." – P4

Here again we can see an iterative prototyping process, however in this participant's case the amount of iterations necessary was increased, and this increase was often due to the misinterpretation of the designers' intent.

The results of the interviews helped guide our high-level goals when designing PaintBoard. We found that participants could benefit from quick prototyping tools, as so much time is spent experimenting in the design phase. According to our participants, this time is not spent exploring, it is spent mostly on planning, programming, and iterations necessary because of miscommunication between designers and developers. Thus, a primary focus of PaintBoard become allowing designers to create and interact with multiple behaviours in a span of minutes. The finding also grounds our initial rapid prototyping motivation in the needs of real users. Participants also indicated issues with collaboration, and that they could benefit from better ways of communication. Exploration of this idea led us to the painting metaphor: painting is visual, and painted storyboards could be understood and discussed by both developers and designers. This approach further benefits from existing knowledge that suggests that visual tools improve communication [5,18].

3.2. Programming Workshop

We conducted a programming workshop to explore approaches currently used by developers to implement interactive behaviours. This is useful for our work for two reasons: we should consider if our interface should accommodate certain ways of framing behavior descriptions, and we can learn from how people programmatically analyze in-game situations, informing our algorithm design by implementing similar calculations. To this end we asked 26 fourth-year undergraduate Computer Science students in a Human-Computer Interaction class (at the University of Tokyo, spring, 2013) to program a set of behaviors. We used a medieval-theme (common within the role-playing video game genre) as a representative scenario. Participants were provided with a simple graphical game board (that looked similar to PaintBoard, see Figure 3.1) and a Java programming interface that allowed students to programmatically query areas on the board for its contents, and

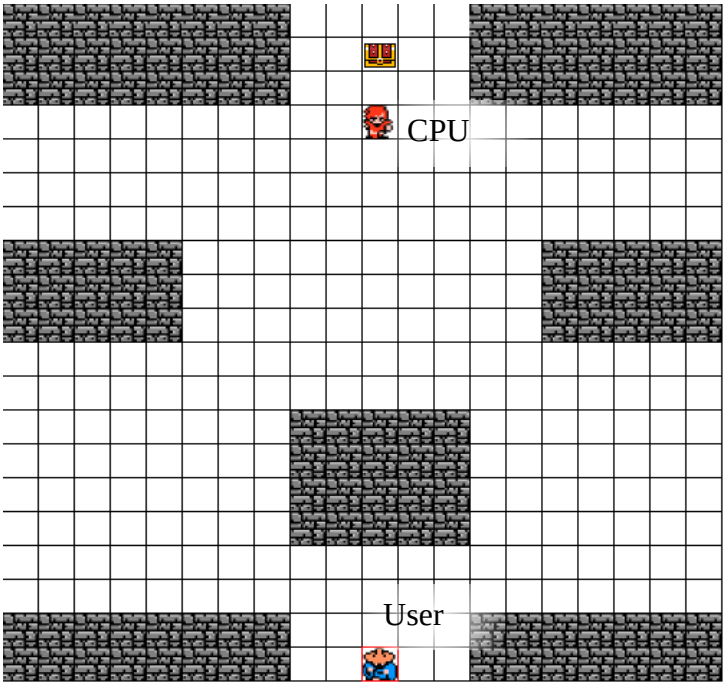

Figure 3.1: The scenario provided to the programmers of the programming workshop. The grey stone areas are not traversable (characters cannot move onto the same space), and the programmers were told the user and computer characters cannot see through the stones either. The programmers created behaviours for the red character (CPU) while users controlled the blue character (User) with the arrow keys on the keyboard.

to provide the next move for the computer-controlled character. Students were tasked with creating three behaviors each, and were encouraged to develop their own original behaviors. We provided "follow the user" "protect treasure from the user" and "escape from the user" as examples.

We received 78 unique behaviour implementations that we categorized into 19 distinct types of behaviours through open-coding based techniques [2]. The three most common of these were our suggested "follow the user" (24 participants), "protect treasure" (18), and "escape from the user" (13) behaviors. The remaining behaviors had less overlap (16 types over 23 implementations) and can be seen in Table 3.1. Common behaviour types (such as our top three), however, had significant variation. For example, some "follow the user" implementations would stay close behind the user, others would walk side by side, and yet others would follow from a distance. This suggests that PaintBoard should accommodate such variation, as opposed to, for instance, providing premade behaviours.

Our post-workshop exploration of the developers' behaviour implementations illuminated some programming strategies that our participants used to define their behaviors. We iteratively tagged implementation strategies in all behaviours, which allowed us to identify common techniques and approaches. Participants consistently used two techniques. The first a small set of commonly calculated quantities, such as the characters' relative positions, to decide on how the computer character should interact with the user. We explored such calculations in our development of our PaintBoard algorithm (Section 4.3). Another common technique was to specify goal locations or points of interest, for example, a treasure chest to "guard," a "hideout" to run to, or the more abstract "staying in close proximity to the user." We included the ability to specify such areas in PaintBoard's interface by enabling users to paint goal areas.

3.3. Conclusion

Our initial investigation informed us not only of the variety of interactive behaviours possible, but also of some of the common techniques used by our developers to implement those behaviours. Initial interviews with professional game designers informed us that iterative prototyping is central to early explorative work, but this process is often hampered by the slow nature of programming or difficulties communicating interactive behaviour ideas with their colleagues. Our behaviour programming workshop told us that a behaviour was envisioned differently by different authors; PaintBoard will need to accommodate not only a variety of behaviours, but allow variations of those behaviours. By exploring our participants' implementation methods, we found that framing behaviours in terms of goal states (be hidden, stay in front of user, etc.) was common, which motivated us to enable the user to define goal areas with our painting-interface, and to look at the context of the goal areas themselves (such as identifying if the goal is to be hidden, rather than to simply stand behind a certain wall). It also revealed calculations frequently used to define the state of a behaviour, such as if the character can be seen, and how far it is from the user. Thus, we explored these calculations for our algorithm's method of analyzing the painted storyboard input detailed in Section 4.3.

Behaviour Name	Description of the behaviour	N
Follow	Follow the player	24
Protect treasure	Protect a point in the environment marked by a treasure box	18
Escape	Run away from the player	13
Lead	Lead the player towards a point in the environment marked by a treasure box	4
Block Path	Block the movement of the player by standing in front of them	3
Hide	Stay out of the sight of the player	2
Imitate	Copy the movement of the player	2
Mirror	Copy the movement of the player in reverse	2
Hideouts	Hide from the player by travelling between certain points in the environment	1
Drive away	Chase the player away from a certain point in the environment	1
Path	Follow a preset path	1
Panic	Run around the area in quick, large movements	1
Dog	Run back and forth between the player and a point in the environment	1
Boo	Move towards the player when the player is not facing the computer character	1
Move along wall	Sneak around the area by staying close to the walls	1
Steal treasure	Move the treasure out of the environment	1
Safe Steal	Steal treasure, but Escape if seen	1
Antisocial	Stay away from the player, preferring to be out of sight, but moving far is unnecessary	1
Possessive	Stay near a treasure chest whenever the player is near	1

Table 3.1: Our classification of the 78 behaviour implementations. N is the number of implementations in the classification. Implementations were given exactly one classification.

4. PaintBoard: Interface and Algorithm

Our PaintBoard prototype enables people to prototype interactive behaviours by digitally painting a scene (as seen in Figure 4.1). While designing and developing PaintBoard, we had two challenges: create an interface that enables behaviour authors to quickly describe their desired behaviours, and create an algorithm that takes the behaviour described by the authors with the interface and produces an interactive behaviour instance. For the interface, we took advantage of the benefits of low-fidelity prototyping techniques, discussed in Section 2 [18,27,34], and developed a novel approach that enables users to quickly define a behaviour by digitally painting simple scenes, similar to sketching. Our approach further draws from the practice of storyboarding to enable authors to design behaviours piece-wise, to focus on one part of the behaviour at a time. In PaintBoard, this piece-wise creation is realized through enabling authors to digitally paint one or more static *snapshots* that make up a storyboard (Figure 4.1b), where the overall storyboard represents the entire behaviour.

PaintBoard's algorithm has to interpret a user-painted storyboard and generate an interactive behaviour result. The behaviour generation algorithm analyzes an author-painted storyboard by extracting information from the storyboards that is relevant to the behaviour; in order to identify

which elements of a behaviour storyboard provides this important information, we draw on the results from our programming workshop (Section 3.2). We then use the extracted information to learn how the author painted the behaviour. During interaction as the user moves their character, we extract information about the real-time situation that enables us to use what we learned from the authored storyboard to predict how the author would have painted the current scene. From this painting, we decide the next move for the computer character. Below we address the interface design in detail before covering the PaintBoard algorithm.

4.1. PaintBoard Interaction Flow

To facilitate rapid and iterative prototyping, it is important to enable users to not only quickly create behaviours, but to continuously modify them and easily interact with the prototype behaviour. In PaintBoard, behaviour authors first create a game scene for the behaviour to take place in by placing bricks that can represent buildings and walls, and by placing the user and computer controlled characters in that environment. The authors then paint a behaviour. When done, they can interact with the behaviour by controlling the user character and seeing how the computer character reacts with the behaviour generated by our system from the author's painted storyboard. The author can then update the behaviour, or try a new behaviour, iterating the author-test-update cycle until they are satisfied.

In PaintBoard, authors paint each snapshot of the behaviour storyboard by choosing a colour from the palette (Figure 4.1c) and then by clicking-and-dragging on screen in the game environment (Figure 4.1a), similar to common computer painting applications. Our paint colours were chosen

based on trends observed in our programming study (Section 3.2); we discovered that programmers often defined areas that were desirable (goals) or undesirable (areas to be avoided). Thus, we chose three paint colours: red paint to represent areas that the computer character should avoid, gold paint to represent areas the computer character should move towards, and an unpainted "colour" to represent areas that are neither desirable, nor undesirable. Painting is described in more detail in Section 4.2.

While painting, users can interact with their behaviour by pressing a single button ("play") (Figure 4.1e), and the system, in real-time (without delay), compiles the behaviour and generates a result that the users can interact with using the keyboard controls (arrow keys). At any time, the author can modify the behaviour by pressing "stop" (Figure 4.1e), which stops the computer-controlled player's movements. The author can then edit any of the paintings in the storyboard, or create a

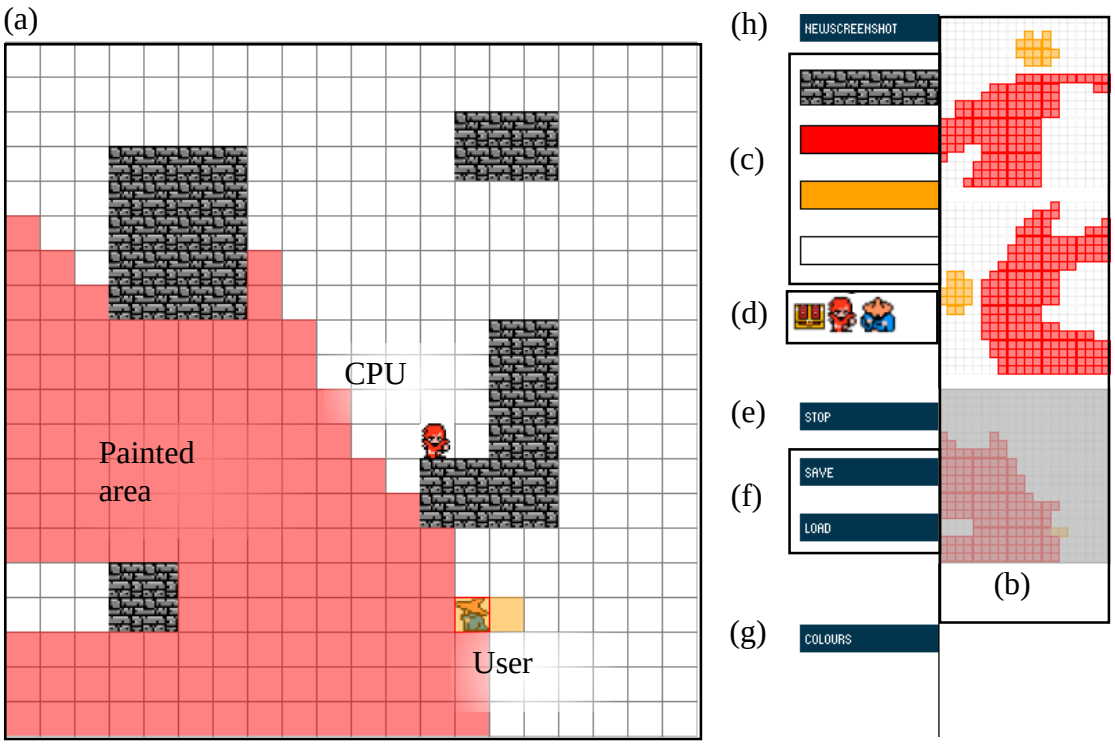

Figure 4.1: The PaintBoard interface: (a) sandbox area, (b) storyboard, (c) paint palette, (d) point of interest (chest) and characters, (e) play and pause, (f) save and load behaviour, (g) debug mode, (h) new storyboard frame.

new snapshot to add a new part of the behaviour. While interacting with the behaviour ("test mode"), if the computer character moves in an unintended way, the author can stop the interaction and use the current problematic scenario as a new snapshot by painting it to become a new storyboard frame, clarifying how the behaviour should act. Thus the author can rapidly change between modifying their storyboard and interacting with their results.

PaintBoard has a debug mode that is identical to test mode except that it provides real-time visual feedback of the computer-controlled character's behaviour in the current situation: PaintBoard paints squares to indicate where—in the given situation—it believes the character should and should not go, and does this with the same red, gold, and white paints authors use to describe behaviours. For example, given the snapshots in Figure 4.2 as the storyboard input, Figure 4.1a is actually the debug-mode painting—a visual representation of what PaintBoard has learned. Authors can use this to gain insight into how PaintBoard is interpreting the storyboard, and can modify their storyboard or even paint the debug frame itself to update the behaviour.

4.1.1. Interaction and Interface Design Rationale

Enabling rapid and iterative prototyping of ideas was a major interface priority, as it has been shown to aid creativity and exploration [27,34,36]. Our PaintBoard implementation helps users to explore their current behaviour prototype by enabling them to interact with the behaviour resulting from the painting and supports iteration by enabling authors to modify existing snapshots in the behaviour storyboard, or even introduce new snapshots. Finally, the storyboard technique was chosen to help authors design multifaceted behaviours piece-by-piece, lowering the complexity required to handle it all at once (a problem found in prior work [41]). Storyboarding also helps

prototype behaviours as authors can iterate on a single part of a behaviour, such as "guard the treasure if the player is nearby," before expanding the storyboard to new situations (e.g. "don't leave the treasure room when chasing the player"). Finally, PaintBoard's behaviour authoring does not require any computer programing or logic definition, which helps speed up prototyping by avoiding the need to specify precise and detailed information.

An important design decision was for behaviour authors to be able to iterate a behaviour idea (create, test, modify, repeat) from a single interface. PaintBoard enables authors to paint behaviours *in-situ* (as in [21,33]), that is, they create behaviours in the same environment where the behaviour will be interacting with the player. This enables authors to bypass conceptual translations required when moving from an authoring to a testing medium (e.g., moving from visual programming to a game), helping authors to visualize the final result [3] and thus more easily create behaviours.

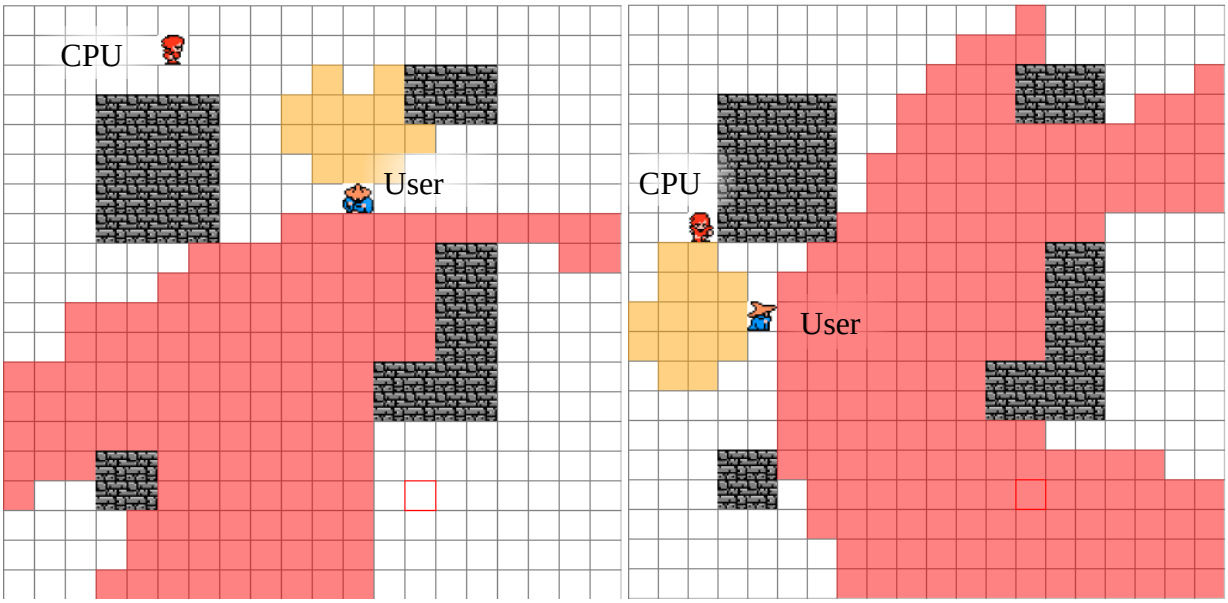

Figure 4.2: A sample two-part storyboard for a behaviour of a computer character (CPU) that sneaks up on the user character (User). In this case, the computer character should not enter the sights of the user character, (red squares) and should stay close to and behind the user (gold).

4.2. User Interface

For each snapshot in a storyboard, authors paint a scene (Figure 4.1a) to represent one aspect of the desired behaviour. The scene can include a user character and a computer character, some environment features such as walls, and important entities. Important entities, or *points of interest*, were a common element of behaviours we discovered during our programming workshop in Section 3.2. Points of interest are objects or locations in the game world that the behaviour revolves around, such as treasure chests or hideouts. For example, a snapshot may have a treasure chest in a room, with the computer character standing between the user character and the chest, which might represent the computer character guarding a treasure.

In order to simplify the problem for our initial exploration, both in terms of authoring and behaviour generation, we had each snapshot be painted in a 20x20 2D grid for our authoring interface, where the author paints snapshots from an overhead view (Figure 4.1a). Authors can drag objects (from Figure 4.1d), including both the computer and user characters and points of interest (treasure chest), onto the grid, and position them as desired. In order to allow faster creation of environments, environment walls (the bricks from Figure 4.1c), are placed with a click-and-drag action, putting a brick in every grid cell along the path of the mouse cursor. To focus on the initial problem of one-on-one interaction, only one computer controlled and one user character, as well as one point of interest is allowed at a time in this version of PaintBoard; co-ordinating multi-agent behaviours remains important future work.

We designed our painting technique to be similar to common computer painting applications to make PaintBoard more approachable: authors select a colour from a colour palette (Figure 4.1c),

and then paint on the grid by clicking and dragging the mouse, filling grid cells along the path of the mouse cursor with the colour; as we aimed to keep authoring simple for authors, cells can only have one colour at a time, and painting over an already painted cell will replace the old colour with the current one.

Our choice of paints was a result of our findings in our programming workshop (Section 3.2), where we found programmers often defined types of areas such as goals for the computer controlled character (e.g. "go anywhere near the player character"), or types of areas the computer character should avoid (e.g. "avoid anywhere the player can see"). In our interface, red paint denotes areas where the computer character should *not* go; for example, when painting a "sneak up to the user character" behaviour all grid cells that the user character can see should be red because a sneaking character does not want to be seen (Figure 4.2). Gold paint indicates *goal* areas: where the computer character wants to go (for example, in Figure 4.2, the sneaking character wants to get close and behind the user character). Unpainted (white) squares are neutral and the character neither avoids nor tends toward them. They can be thought of as places the computer character can use freely to get to their goal. In our interface, every cell, by default, starts unpainted, but painted cells can be made unpainted again by using the white paint, seen in Figure 4.1c. In other words, the computer character should try to go toward gold areas, while passing through unpainted areas and avoiding red ones.

Goal areas and points of interest are similar, but conceptually different entities in PaintBoard. Goal areas can be abstract (not a specific point in the environment) and dynamic areas the computer character wants to go to, and are based on the user and computer characters' current configuration (e.g., "behind the user" or "block the path to the treasure"). Points of interest, however, are static

objects in the environment that are important to the interactive behaviour (e.g. hideouts, guarded items). In PaintBoard, not all behaviours may have a point of interest, but we assume they will all have goal areas.

In addition to "sneak" (Figure 4.2), other examples enabled by PaintBoard similar to those observed in our programming workshop include "follow the user character," by having a goal behind the user character, and "guard the treasure" by defining the treasure chest as a point of interest, having the goal (gold paint) between the user and the chest, and painting red squares far from the chest to keep the computer close. For variants on "follow," some examples include keeping the computer character behind or to the side by painting the respective areas gold, or within a radius by painting far-away squares red.

In our programmer workshop (Section 3.2), we found that the developers often framed the goal and avoid areas in terms of various quantities their behaviour-generating programs calculated while interacting with the user character, such as distance from the user character, or if the characters can see each other. PaintBoard enables authors to define behaviours using similar techniques, but without explicit calculations, by painting areas that have the correct properties: "avoid line of sight" can be defined simply by painting red where a character can see (Figure 4.2); staying close to the user can be defined by painting areas around the user character gold. We believe this allows enables authors to create behaviours on an intuitive level: instead of authoring in an abstract way with calculations and computer code, they can author in terms of the environment and character's physical relationships in the context of the game itself (in-situ).

The rest of the sidebar in our interface contains the storyboard overview, where users can scroll through and select snapshots to edit them (Figure 4.1b), save and load buttons (Figure 4.1f), and a

button to add a new snapshot to the storyboard (Figure 4.1h). In addition, there are buttons for changing between painting and testing modes (Figure 4.1e), and for entering a debug mode (Figure 4.1g).

4.3. Algorithm

The goal of the PaintBoard algorithm is to analyze an author's painted storyboard and to generate an interactive character behaviour that matches the qualities given in the storyboard. To simplify the algorithm, we redefine the problem to: given a run-time situation (including a set of walls, the computer and user character's position, etc.), PaintBoard must generate the computer controlled character's next movement based on that situation. However, this run-time situation is very unlikely to be represented directly in the storyboard: to encourage rapid prototyping, we envision storyboards being short and only covering a few exact situations.

Our solution is to first generate an *approximation snapshot* that estimates how the current situation would have been painted by the designer. The approximation snapshot can be thought of as a snapshot that is an interpolation between the existing snapshots in the painted storyboard. For example, given the two snapshots as input from Figure 4.2, PaintBoard generated the painting in Figure 4.1a to have similar characteristics (stay out of line of sight, etc.). Following, we use the painted approximation snapshot to generate the computer character's next movement. This process is real-time: to inform the next computer character movement, a new approximation snapshot is generated for the new situation each time the characters move.

4.4. Generating Approximation Snapshots

During real-time interaction, approximation snapshots need to be generated for a new situation not given in the storyboard, while staying similar to the properties of the user-painted snapshots. Specifically, we need to determine, for each cell in the game grid, if the author would have painted that cell red, gold, or left it unpainted. The result of painting all cells is a new fully-painted approximation snapshot that can be used to generate the output behaviour.

Our approach is to use supervised machine learning to estimate what colours each cell should be painted in a new approximation snapshot. Supervised machine learning algorithms are generic learning algorithms that, given input training data, create a model that can be used for prediction, called a classifier. The training data is a list of data points that each have a label, called a class, that describes the category of that data point. The trained classifier can be used to predict the class of new data points. For machine learning classifiers, all data points are usually vectors (a list of numbers) of equal dimension (each vector has the same number of elements). Each number in the vector measures some descriptive quality of the data point. For example, a classifier learning about rectangles and squares might take vectors of length three—maybe height, width, area—with all vectors where height and width are equal are labelled as "square," else they are labelled "rectangle." The classifier would, ideally, learn that, given a vector of length three, if the first two values are equal, then the label should be "square." With machine learning classifiers, more training data generally produces a better classification [14]. In PaintBoard, storyboards are assumed to only contain a few entries; to increase the amount of training data available per storyboard, we decided a single data point is a grid cell in our environment (for example, see Figure 4.3), and the possible

classes are the three possible paint colours (red, gold, and unpainted). We use all grid cells to train our classifier. One challenge of PaintBoard was how to represent a grid cell as a data point.

To make a grid cell into a data point, we needed to select representative features (descriptive numbers and calculations) that capture the appropriate characteristics of a grid cell. This is non-trivial, and we had to develop our own domain-specific features given the lack of prior work. For PaintBoard we used features identified through our programmer study (Section 3.2). Note that these features represent what determines what colour a cell should be painted. We call these *state features*, and they take into account the context (the state) of the behaviour; for example, there is a state feature that quantifies how far away the two characters are from each other. The state features are detailed below in Figure 4.3 and Section 4.4.1. Development of appropriate domain-specific features is a common challenge and has been done for other applications such as for recreating body language from a demonstration [12,31], non-interactive behaviours for industrial robots [16], and nature simulation [32].

For our machine learning algorithm, we employed a Support Vector Machine (SVM) [8] as it is known to be a standard, fast classifier. In PaintBoard, we train a new SVM for an authored storyboard immediately when the author clicks the play button (Figure 4.1e). We transform the storyboard into a set of feature vectors: these vectors are calculated for every cell in every snapshot, and each contain the measurements of our state features for that cell (see Figure 4.3). Each feature vector is labelled with the colour it was painted by the author (red, gold, or unpainted). All labelled vectors are given to the machine learning algorithm which outputs a trained classifier that can be used to predict the label of other feature vectors.

Once we obtain the classifier, we can create the approximation snapshot for the current scenario. Starting with an unpainted snapshot, we calculate the state features of each grid cell. One by one, we give these state features (a data vector) to the trained classifier, which returns the colour it believes the author is likely to have painted that cell. The trained classifier would ideally label new unpainted cells of the approximation snapshot to match characteristics (the features) of the training data (for example, as seen in Figure 4.1, where the classifier predicted red paint where the user character can see, just like in the storyboard).

4.4.1. State Features

This section describes the complete list of state features used in the current version of PaintBoard. All features, unless otherwise noted, are calculated for each grid cell.

Position relative to the user character with respect to screen axis: The cell's position in relation to the user character, in the screen's coordinate system, for example, in Figure 4.3 the bold cell is two to the left and three above the user character. This captures the importance of the screen's position (e.g., stay on the left or right side of the environment).

Position relative to the user character with respect to its look direction: The cell's and point of interest's position in relation to where the user character is looking, for example, in Figure 4.3 the bold cell is two cells in front of and 3 cells to the right of the user character. This captures the importance of a cell's position from the user character's point of view (e.g., stay behind them).

Position relative to the user character's relation to the point of interest: We calculate a coordinate system rooted at the user and oriented to the point of interest, and use the cell's position in that space. For example, in Figure 4.3 the bold cell is 2.6 cells behind and 2.5 to the left of the user and chest. This captures the context of the point of interest (e.g., do not go between the user and the chest, stay close to the chest). This is not used when there is no point of interest.

Visibility: A scalar representation of how well the user character can see that grid cell: we cast rays from the user to the cell and that cell's neighbours to calculate visibility, with those blocked by walls not counted. For example, the bold square in Figure 4.3 has visibility 0.6 (6/9). This captures line of sight information (e.g., how visible cells are to the user character), and the non-binary classification enables the computer character to capture the difference between being partially and fully seen.

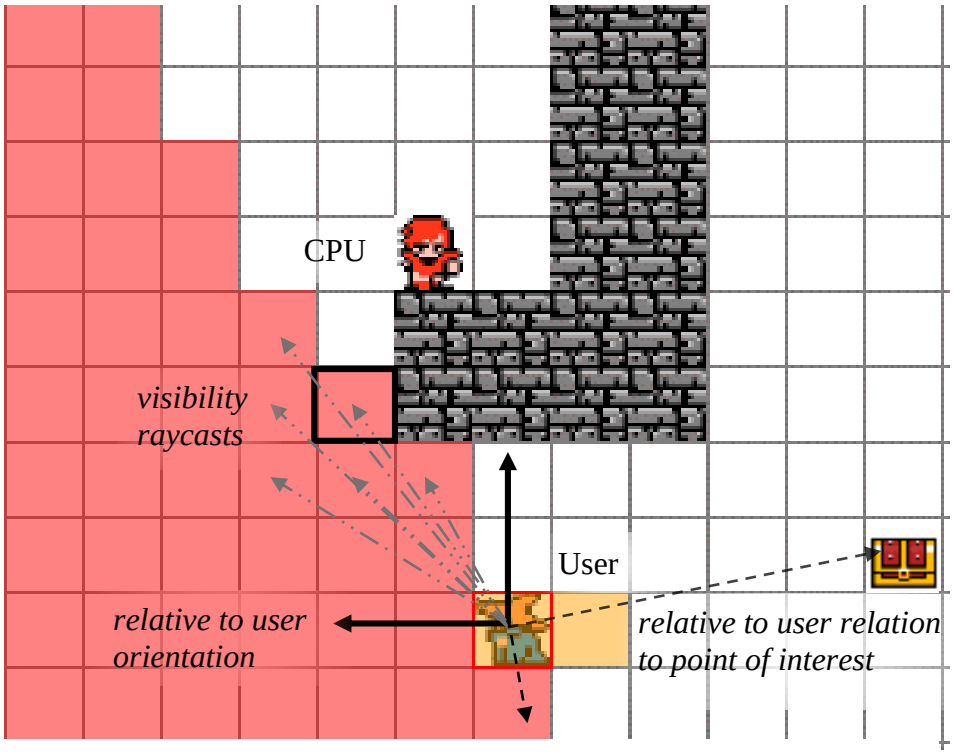

Figure 4.3: How state features are calculated, e.g., for the bolded cell.

Absolute distance from the user character and the point of interest: The Euclidean distances from the cell to both the user character and the point of interest. For example, in Figure 4.3 the bold square is 3.6 and 7.3 from the user and point of interest respectively. In combination with the relative features above, this helps emphasize proximity (e.g., how close to be to the user).

4.5. Using an Approximation Snapshot to Generate the Behaviour

Given an approximation snapshot for the current real-time situation, our challenge was to generate the next move for the computer controlled character. Our approach was to simply move the computer character toward the closest goal space (gold paint), while not walking through walls, and avoiding red spaces if possible. To find the closest goal space, the algorithm does a breadth-first search in polar coordinates, spiraling out from the computer character, where walls are considered impassable. Red cells may not be completely avoidable, for example if all other paths are blocked. We address this by penalizing red cells by giving them a distance of four when calculating the nearest gold square, making longer paths with no red still favorable over shorter paths that cross red, but short paths across red areas are favorable over significantly longer alternates. For the case when the character is stuck with large red areas between it and the goal, we added a path-length threshold so the character simply would wait in safety rather than traversing large red areas. These values were chosen based on experimentation by a researcher on the project, but we stress these values are only for proof-of-concept, and more formal studies will be necessary to explore how this method of path-planning could be changed, or improved.

4.6. Implementation Details

We used the Java LibSVM library (version 2.89) [7] with its default settings for the SVM implementation. The logic was programmed in Java (1.7.0) using the Processing framework (version 2.0.1)[3] and the ControlP5 library (version 2.0.4)[4] provided the graphical user interface functionality such as the buttons we used in our interface.

4.7. Summary

In this section we presented our PaintBoard implementation, which included both the interaction and algorithmic design. Our interface, inspired by our initial investigations, allows authors to paint environments in a series of snapshots in a storyboard, taking advantage of the benefits of low-fidelity prototyping techniques. The interface also supports iterative prototyping by enabling authors to edit and test their behaviour in the same environment (*in-situ* design). PaintBoard's algorithm generates a behaviour by analyzing a storyboard and then, in real-time, predicting how the author would paint the current scene and moving the character according to the painting. The algorithm analyzes both the author's storyboard and real-time situations using *state features*, quantities that describe the context of an area in the environment, which were derived from the results of our programmer workshop.

[3] http://www.processing.org/

[4] http://www.sojamo.de/libraries/controlP5

5. Evaluation of the Painting-Storyboards Technique

We conducted a proof-of-concept workshop to explore reactions to our PaintBoard approach and interaction design by potential end-users. For participants, we recruited three professional and two hobbyist game developers. The workshop took 1.5 hours and comprised of a tutorial, an unstructured authoring phase, and a questionnaire period. The tutorial was 15 minutes in duration, and taught participants about how to use PaintBoard to prototype interactive characters. Afterwards, they were given one hour to freely create any behaviors they wished, ask questions, and discuss with other participants. The workshop ended with a 15 minute questionnaire that asked participants about their experiences with PaintBoard. Although each participant worked independently, the atmosphere was friendly and collaborative, and people were having spontaneous discussions about their experiences. Notes were taken throughout the workshop by the researcher at the workshop.

Participants were asked to save their storyboards through the PaintBoard functionality for later inspection. In addition, we performed broad qualitative analysis on the notes and questionnaire

answers by iteratively tagging answers with keywords describing the statements. Through this, we identified themes and insights of our participants on topics such as viability of painting as a behaviour authoring technique, or participant-suggested ways that PaintBoard could be leveraged in real-world situations. See Appendix B for the materials used in this study, including the ethics amendment approval certificate from the Joint-Faculty Research Ethics Board.

5.1. Results

Overall, participants were able to use PaintBoard to quickly and successfully prototype a range of interactive behaviors (create 3 or more behaviours in the one hour). These included “follow the user character,” “hide,” “obstruct the user character,” “guard an area,” and “sneak.” This was achieved with a relaxed-pace 15 minute tutorial, suggesting that our painting and storyboarding approach and implementation was approachable to new users.

Participant responses in our questionnaire data also contained support for PaintBoard’s success; they reported that PaintBoard would be useful for planning and prototyping ideas:

> *In its current state, could be handy for prototyping and visualizing scenarios. - P3*

> *I would use this as a prototyping tool to make quick behaviors that I would then implement with code - P2*

and for communicating with others:

Easy to visually show others simple behavior that can be expanded to more complex situations. - P5

Some participants noted that it may be useful for team members with less technical expertise:

I'm not sure if it'll be useful in my workflow (yet), but I think it'll be great for designers - P1

The previous quotes highlight the goals we identified in our exploratory investigations (Section 3.1): facilitating prototyping and communication between designers and developers. This indicates that our results matched our motivation. Participants found the quick, visual, and interactive nature of PaintBoard to be important and useful for behaviour prototyping. In addition, as PaintBoard requires no coding knowledge, it may enable two-way communication during prototyping as both designers and developers could modify the interactive behaviors to enhance discussions.

Participants praised the benefits of PaintBoard's iterative nature, noting that it matches their existing workflows:

I like the iterative design process. Games tend to follow on iterative design, so this fits nicely. - P1

Even though our participants were experienced programmers, they were very receptive to the use of painting, rather than writing computer code, in the behavior design process:

I think the abstraction of the concepts to be very easy to understand … as well as the ability to alter states during play, and ability to watch the goal and avoid state change - P4

All participants also felt the performance of the test mode, where PaintBoard generates the behaviour from the storyboard and moves the computer character according to the result, was

reasonable. However, some did show concern over PaintBoard's ability to scale up to more complex behaviors:

> *It's a bit hard to convey a behavior sometimes, but maybe that doesn't need to be a goal. It seems to work with simpler behaviors and I think it can be used as such usefully - P1*

We received several examples where the painted storyboard was very clear and descriptive from a person's perspective, but the resulting behaviors were not generated successfully. However, while this is a failure of the current learning algorithm, we believe that this is a success for the painting interface: it illustrates the ability to represent and communicate a desired interactive behaviour through our storyboards. See Figure 5.1, a storyboard produced by a participant in our workshop: it has easy-to-understand snapshots of specific behavior aspects and the overall storyboard clearly describes a complete behavior, but the generated behavior usually predicted only unpainted cells – thus while there are algorithmic problems still to be solved, the interaction paradigm itself was successful in our case.

While PaintBoard often properly identified large areas of color, thin or highly mixed areas of color often disappeared upon testing the behavior (e.g. Figure 5.2):

> *Established "safe zones" or "goal zones" to attract the "passive" NPC. The system would "forget" the yellow zones even despite consistency [across the sketches]. - P3*

> *I wanted to create a behavior that had the NPC directly between the player and the chest… I did not get the expected results - P2 (see Figure 5.2)*

Another interesting trend was how the developers spotted our use of state features even though the underlying implementation (i.e. which state features we used) were not explained in the workshop:

[I easily created] follow and hide. Few instructions required. Uses all of the data (location and direction) being processed—P5

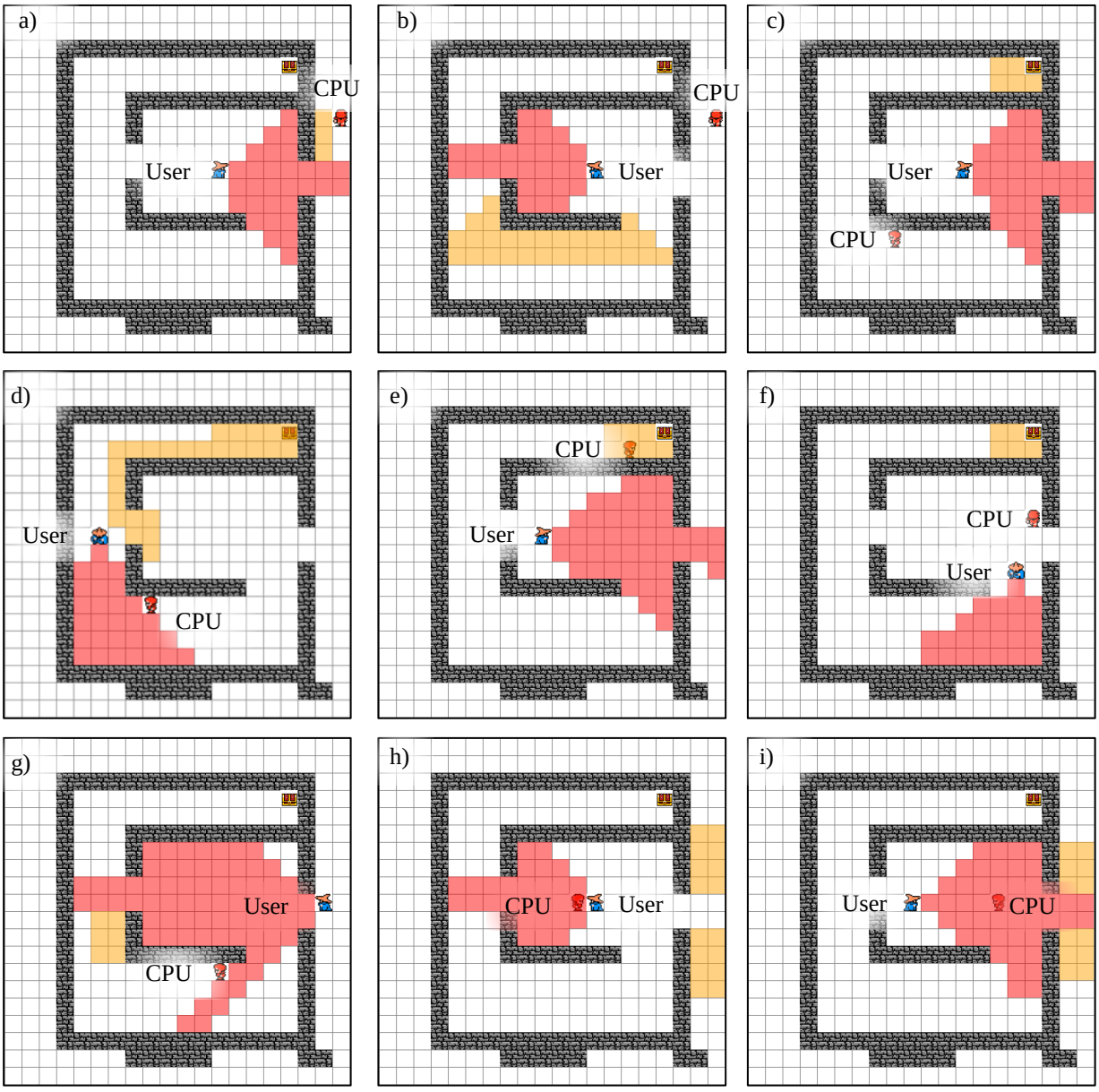

Figure 5.1: A storyboard authored by a participant during our workshop, showing how a computer character should sneak around a user to get treasure. (a) hide by the only entrance to the room (b)when the player is not looking, sneak into the room and stay out of sight (c) when the player is not looking at the inner hallway, run to the treasure (d) if the player is in the hallway, sneak around the other way (e) when at the treasure, stay there, out of sight (f) take the open route to the treasure, but in a different context than d (g) if the player is watching both hallways, get as close to the treasure as possible (h) if spotted by the player, run out of the room and (i) another example, similar to h.

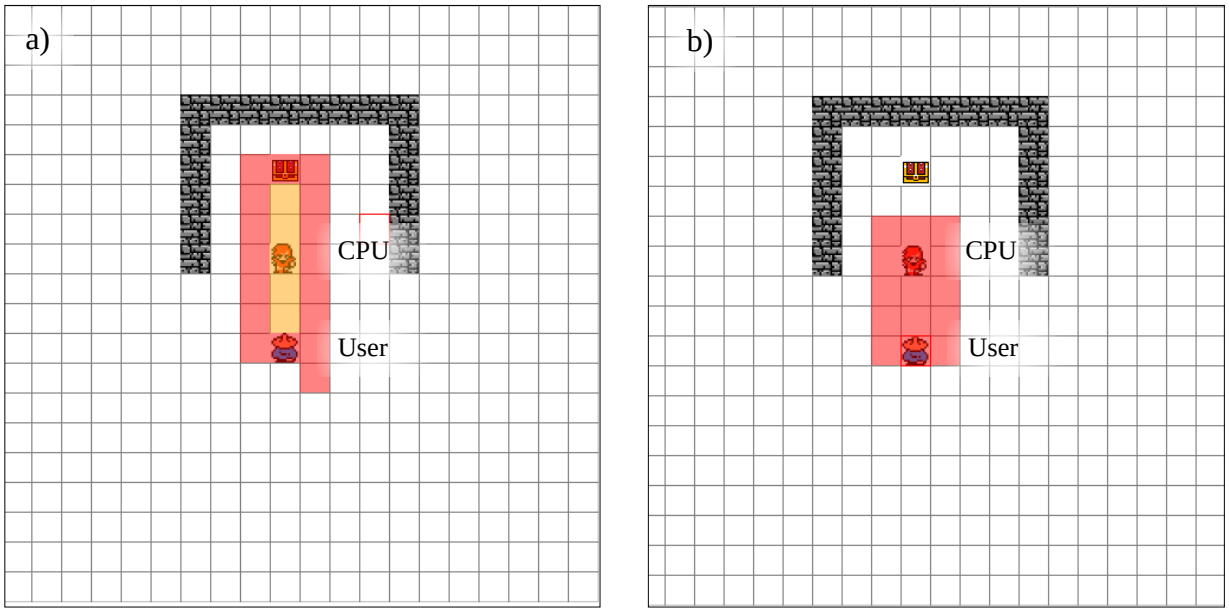

Figure 5.2a: A single-snapshot storyboard from our workshop for a behavior that attempts to teach the computer character to stay between the user and the treasure.

Figure 5.3b: The synthetic snapshot, shown through debug mode. No gold squares are generated in the same context.

Though location and direction were not actually all of our state features, the developers did not know that we were using LibSVM, or how each state feature was calculated. However, our observations suggest that they could still reason about what PaintBoard was doing in intuitive terms that made sense to them.

5.2. Discussion

PaintBoard was successful in its goal of being an initial attempt at enabling prototyping of interactive behaviors through painting and storyboarding; participant feedback supported that we addressed some of the issues raised in our preliminary studies, such as supporting communication and enabling rapid iteration (e.g., through the iterative and in-situ design). In addition to being an

approach accessible to people with programming skills (such as those in this workshop), we aimed for PaintBoard to be useful even for those without any programming experience at all.

We received feedback specifically regarding our algorithm. Some quotes highlighted that participants discovered our state features; the implementation came through in a transparent way, and this suggests that our selection of state features may cover some of the ways developers think about behaviours. Although some behaviours were not able to be generated successfully, this was generally a result of the algorithm and not the painting approach itself, as the storyboards themselves (e.g. Figure 5.1) were clear.

In addition to reflecting on the potential benefits of PaintBoard, participants described specific functionality that they believed could improve PaintBoard. For example, participants requested the addition of story branches, where a condition indicated in a snapshot may lead to a new set of snapshots. This could easily fit within the PaintBoard storyboard interaction, but would require new algorithmic solutions. Participants also suggested adding the ability to make hard rules about the environment, for example, to mark specific squares in the environment which should always be avoided. Another paint related suggestion was the ability to weigh painted cells, where some are more important than others (e.g. prefer not being seen over reaching the treasure). For example, in Figure 5.1 there are many goals in different situations (if the character is seen, if a path to the treasure is open, etc.), however, the developer wanted the computer character to prefer not being seen over reaching the treasure. While the ability to specify such details would give more creative power and control to a PaintBoard user, such features should be added with careful consideration of PaintBoard's of behaviour authoring speed and simple interaction flow, else they may slow down PaintBoard's rapid and iterative nature.

6. Evaluating the Accuracy of Generated Behaviours

For our proof of concept interface evaluation (Section 5), we selected a standard learning classifier and achieved reasonable behaviour generation, according to our initial user study. Unfortunately, to our knowledge, there is no previous work on end-user authoring of interactive behaviors to which a comparison can be made. To provide a quantitative baseline for future work and to understand how our choices of machine learning algorithm and feature set affect the quality of the behaviour generated by PaintBoard, we performed an analysis of behaviours generated by different variations of our algorithm. We modified our algorithm on two dimensions: classifier, and feature set used to train the classifier. This evaluation had three components: we developed a dataset large enough to facilitate both training and testing of an algorithm and an accuracy metric that could be used to measure the performance of an algorithm, we tested the accuracy of a set of classification algorithm variants on our dataset, and, using the best performing classifier, we explored the accuracy of different combinations of state features.

6.1. Building a Dataset for Evaluation

To build a dataset for training and testing algorithms, we recruited participants and had them paint behaviours with PaintBoard. Participants were asked to make multiple example storyboards of a behaviour so we could train PaintBoard on some examples, and test the resulting behaviour on the remaining examples. It is important that a storyboard does not appear in both sets in order to test each algorithm's ability to generalize a behaviour to situations it has not seen before in the training data.

In our experiment, we trained each algorithm on only one example storyboard (which is made up of at least one painted panel). In other words, we use no more training data than PaintBoard normally gets during normal use. This setup was an explicit decision because our target use case is rapid prototyping and, ideally, a PaintBoard user will paint minimal data (one storyboard example) and test the behaviour in many situations. Additionally, training and testing sets were comprised of storyboards from a single author; we did not, for example, train a "sneak up to the user character" behaviour with one author's storyboard and test on a different author's storyboard. This is because we knew from our programming workshop (Section 3.2) that the same behaviour varies between authors: data from one author is likely to produce a different behaviour than the behaviour produced from another author's data. Thus, to facilitate this, each participant was asked to create many example storyboards of the same behaviour

In our study, participants were asked to create a series of examples for three different behaviours: escape from the user character, chosen as a potentially simple behaviour; sneak up to the user character, chosen to require the use of environmental features; and protect a treasure from the user

character, chosen to force the use of a point of interest. For each of the three behaviours, the participant was instructed to create a series of 10 different storyboard examples, resulting in 30 storyboards (with at least 1 panel per storyboard) from each participant (Figure 6.1). The participants were told that each example storyboard they create (comprised of one or more painted snapshots) should fully define the behaviour. In other words, each storyboard could be used by itself to generate the author's complete desired behaviour. This was emphasized to our users in order for us to be able to use each storyboard example as standalone training data. Additionally,

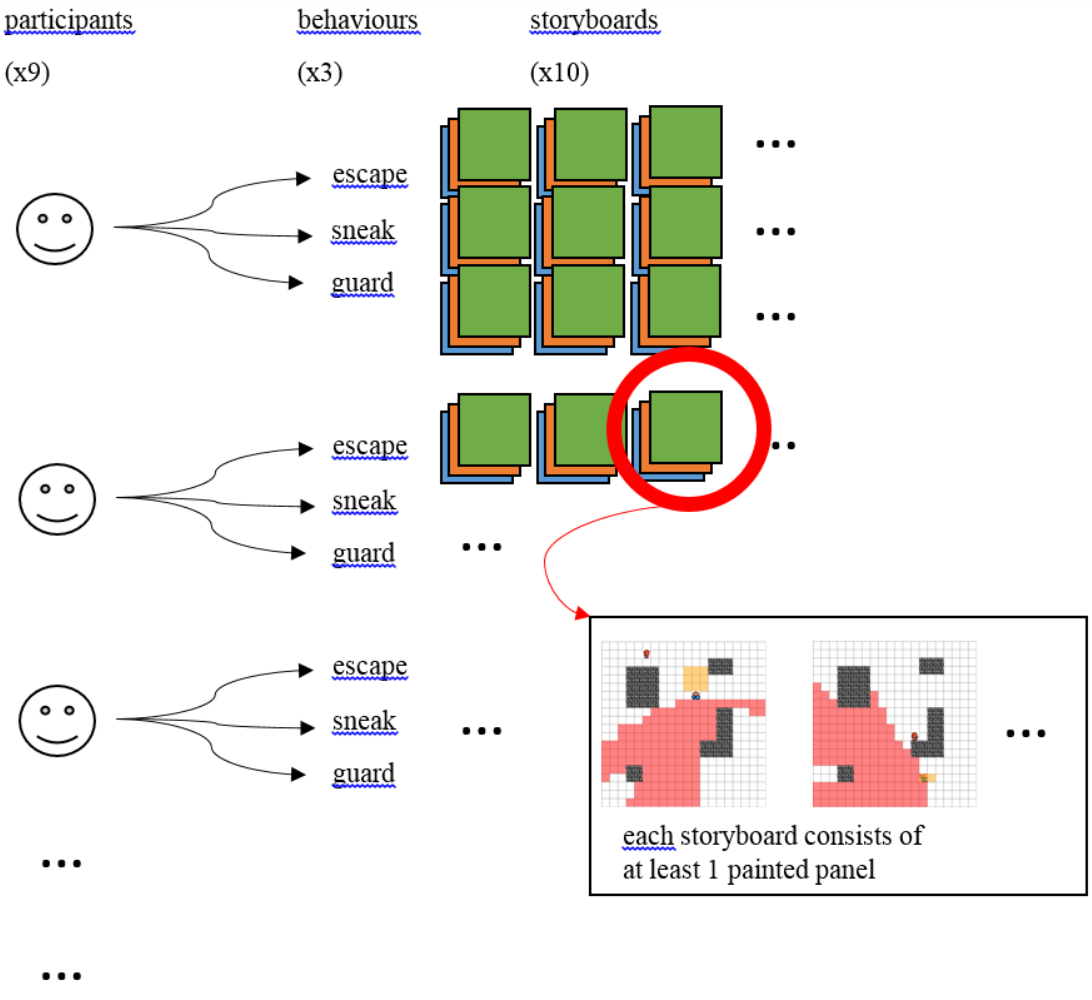

Figure 6.1 How we created our evaluation dataset. Each of nine participants made 10 storyboards with a length at least one panel for each of three behaviours.

having different behaviours enabled us to further investigate if certain algorithms performed better on specific behaviours.

We recruited 9 participants who were 4th year and graduate computer science students. They were given a 10 minute tutorial of how PaintBoard worked, including a demonstration of painting a "follow" behaviour. Participants were informed that their data would be used for evaluating behaviour generation algorithms, though they were not allowed to use PaintBoard's behaviour generation in order to avoid them influencing their authoring approaches to a specific algorithm.

6.2.Comparison of Classifiers

To our knowledge, there is no other previous work to which we could compare PaintBoard, so we instead compared the performance of PaintBoard with five different machine learning classifiers: SVM with a Radial Basis kernel, SVM with a polynomial kernel, K-Nearest Neighbours, Random Forest, and Naïve Bayes classifier. These algorithms are a sample of supervised learning techniques covering several different approaches to machine learning: SVMs are standard, fast classifiers used in a variety of modern applications, K-Nearest Neighbours is a simple but commonly used cluster-based algorithm, Random Forest is a modern tree-based approach, and the Naïve Bayes classifier is based on Bayesian statistics and provides an adequate baseline for most applications [6]. We used the implementations of these algorithms provided by the Java Machine Learning Library [1]. The two SVMs used the libSVM default parameters; k was chosen as 5 for K-Nearest Neighbours as, in initial testing, it had similar performance to higher values (e.g. k=10)

with faster run-time; the tree count for the random forest was set high to 100, based on the evidence suggesting random forests do not often overfit their generated results to their training data [4].

We needed to develop a performance metric that could compare the output from different classifiers—that is, we needed a method to determine which algorithm generated a behaviour closest to the one the author intended to create. This is non-trivial for our problem domain as we must know what the author's intent was, and devise a distance function that measures the similarity of that intent to the trained classifier. We defined the author's intent to be how they painted their storyboard snapshots. Thus we compared the painted storyboard snapshots from the test set to snapshots generated by the classifiers trained on the training-set storyboards; recall that the test set does not contain any data from the storyboard used to train the algorithm. The performance of a classifier was then defined to be the similarity of the author painted snapshots and the snapshots generated by PaintBoard.

Specifically, we trained a classifier with one storyboard (the training set). That classifier then was used to generate synthetic snapshots for situations given in test-set snapshots painted by the same author of the training storyboard. The situation for which we generate the synthetic snapshot is defined by the user and computer character's positions, and the environment (walls and points of interest) in the author-painted snapshots. The synthetic snapshot is generated the same way PaintBoard would normally generate the snapshot if that situation was encountered during real-time testing (Section 4.4).

We compare the authored snapshots with the corresponding synthetic snapshots at the grid-cell level; the paint of each grid cell in the author-painted snapshot is compared to the paint of the corresponding grid-cell in the synthetic snapshot. If the paints are the same colour (gold, red, or

unpainted), it is considered a direct match, otherwise the classifier is considered to have made a mistake. The accuracy of the prediction for one snapshot is defined to be the percent of true positives (direct matches) between the author-painted test snapshot and its synthetic snapshot. The accuracies of all snapshots in a storyboard are averaged together to give one accuracy measurement for that storyboard when using a given algorithm.

For each algorithm, we perform cross-validation [26] to gain a better understanding of its accuracy when trained with different storyboards. For a given participant's behaviour, we trained PaintBoard with only one of the provided 10 example storyboards. Thus, the remaining nine examples are used as test data, and then we measure the accuracy as described above. This is done once for each example storyboard, training a new classifier with the new storyboard and measuring its accuracy against the other nine storyboards for each iteration of the validation. These accuracies are averaged together to give us one accuracy value for that participant's behaviour. The mean accuracy of an algorithm is defined to be the average accuracy over all participants and behaviours.

6.2.1. Analysis

Each of the nine participants created 10 example storyboards for each of three behaviours. By the method described above, we calculated one accuracy metric for each of the three behaviours for each of the nine participants, giving us 27 accuracy values for each algorithm. To analyze these values, we performed an Analysis of Variance (ANOVA) on the data which provides us with an effect size that measures the impact of the choice of algorithm on the accuracy of the resulting behaviour, as well as the ability to detect interaction effects (for example, if the choice of author or behaviour affected the accuracy of the algorithm).

Our participants found PaintBoard fast enough in our workshop, but different algorithms with higher performance may also have slower execution speed. It is important for the algorithms to be quick to maintain fluid authoring and interaction. As such, we also performed an ANOVA on the execution time of each algorithm; for each iteration of k-fold validation, we recorded an execution time. These were averaged to give a time for each behaviour of each participant (27 values for each algorithm). The ANOVA was performed on these values, similar to how we analyzed accuracy.

6.2.2. Results

We present the mean accuracy of each algorithm in Figure 6.2. Errors bars show standard error.

Statistical analysis revealed a main effect of the algorithm on the accuracy of the synthetic snapshot ($F(4, 130) = 11.9$, $p < .001$, $\eta^2 = .284$). Post-hoc tests (with Bonferroni correction) revealed that the radial basis function SVM performed better than Naïve Bayes ($p < .001$), and polynomial SVM ($p < .05$). Comparisons of the radial basis function SVM's performance with the RandomForest and K-Nearest Neighbour algorithms' performance showed no significance.

We grouped accuracies by behaviour type across participants. The behaviour type had a main effect on the accuracy of the synthetic snapshot ($F(2, 132) = 8.0$, $p=.001$, $\eta^2 = .117$). Post-hoc tests (with Bonferroni correction) revealed that "sneak" was more accurate than "escape" or "protect" (10% more) $p < .005$. There was no interaction effect between algorithm and behaviour type on accuracy of the synthetic snapshot.

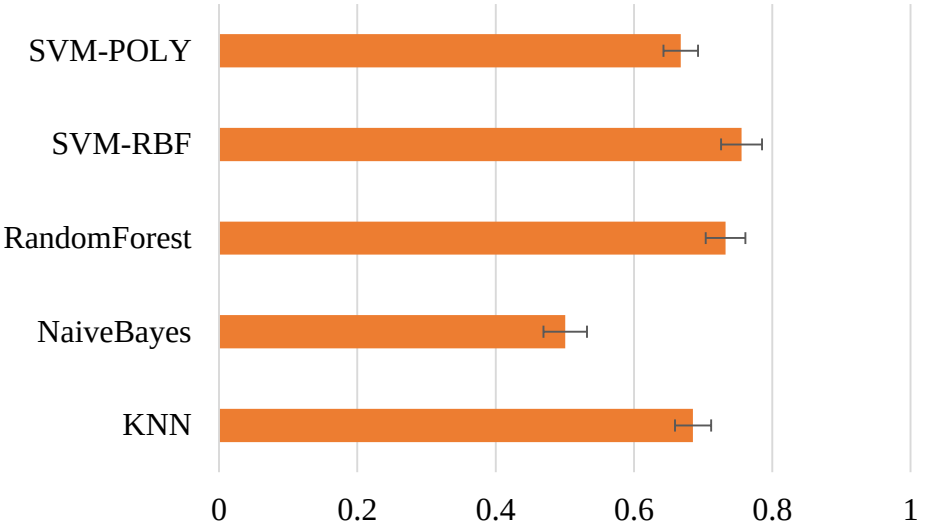

Figure 6.2: Accuracy of each algorithm for our dataset. Error bars are standard error. From the top: Support Vector Machine (SVM) with a polynomial kernel, SVM with a radial basis function kernel, Random Forest, Naïve Bayes, and K-Nearest Neighbours.

One problem with interpreting the above accuracy results is that, in our data, a cell is much more likely to be unpainted (clear) than painted (red or gold); this biases classifiers to give us high accuracy for unpainted cells while possibly lowering the accuracy for the other colors. To provide insight we present a confusion matrix for the radial basis SVM (Table 6.1), showing the average accuracy across all participants and behaviours as a percentage. Each entry can be read as "[entry value] percent of all author-painted [column] cells in the test data were predicted to be [row] by PaintBoard." For example, we can see that, across all participants and behaviours, 76% of gold painted cells were predicted to be unpainted in the synthetic snapshots.

Although our current implementation is sufficiently fast for interactive results, we analyzed execution time as a significantly faster algorithm could be important for future work. There was a main effect of the algorithm on the time taken to run (perform cross-validation) per participant per behaviour, ($F(4, 22) = 19.430$, $p < .001$, $\eta^2 = .649$). Post-hoc tests (with Bonferroni correction) revealed that all algorithms ran at least 650% faster than the polynomial kernel SVM, $p < .001$.

The difference in execution time of the cross-validation with the radial basis function kernel SVM when compared to K-Nearest Neighbours or Random Forest was not shown to be significant. There was no effect of behaviour type and no interaction effect between algorithm and behaviour type on the runtime of the classifiers.

		author-painted colour		
	SVM-RBF	*red*	*unpainted*	*gold*
predicted colour	*red*	0.25	0.09	0.09
	unpainted	0.70	0.85	0.76
	gold	0.05	0.06	0.15

Table 6.1: The confusion matrix for radial basis function SVM. Entries represent the accuracy, ranging from 0 to 1.

6.3. Evaluation of State Features

To investigate the effect of our set of state features on PaintBoard's performance, we briefly explored accuracy variation with subsets of our features. Using only the radial basis kernel SVM (as we found it to be one of the best performing), we performed a naïve greedy feature selection: we measured the accuracy (as above) with each state feature on its own. We picked the feature with the highest mean accuracy across all behaviours and participants. We then measured the accuracy of each feature combined with the selected best feature. This was repeated for all features until no statistically significant improvement was made with the addition of any other feature.

The two selected state features were "*relative to user with respect to user's look direction,"* and *"relative to user's relation to point of interest."* With these features, PaintBoard had a 77.5% accuracy, while it achieved a 77.3% accuracy with all features. This difference was not found to be statistically significant.

6.4. Discussion

Our comparison of multiple algorithms showed our original choice of SVM with a radial basis function kernel was among the fastest of the machine learning techniques we tested. In addition, our workshop (Section 5) suggested that it was fast enough for real-time interactivity. At least, with our algorithmic approach and feature set, we found that there is likely no easy benefit to using similar out-of-the-box machine learning solutions. Even so, PaintBoard sometimes struggled to properly generate a prototype behaviour, despite the storyboards themselves (e.g., Figure 5.1) being clear, suggesting that an alternative, perhaps fundamentally different learning solution may be necessary to improve PaintBoard's behaviour generation accuracy.

As seen by Table 6.1, the largest type of error made by our approach was misclassifying red and gold cells as unpainted. One possible cause is the statistical nature of our machine learning algorithms; due to the large number of unpainted cells typical in our approach, the probability of a cell being similar to unpainted cells becomes higher. Thus, even if all author-painted gold cells are very similar in terms of state features, it is likely that an even larger number of unpainted cells are also similar to those gold cells, making classifiers more likely to paint a cell unpainted. In our case, one possible solution is to carefully balance the data fed into the SVM by clustering the

disproportionately large number of uncolored squares to a representative subset that is similar in size to the other colors. Such a change would benefit from a comparison similar to what we did in this chapter, as different algorithms may react differently to the balanced dataset. Further, many other techniques exist for dealing with imbalanced data, and is an active field of research in machine learning [14]. Changes to PaintBoard's interface may also help because it is possible that, due to all cells initially being unpainted, they are more likely to be left unpainted, as it takes a decision from the user to change this. Despite this limitation in our results, we highlight that labelling a square as unpainted is as equally important as painting it (incorrectly painting them red or gold could create incorrect behaviours), and that the generated behaviours were robust enough for most prototyping, as suggested from the results of our workshop (Section 5).

A key area of potential improvement is our selection of state features. Our greedy feature selection resulted in only two features giving similar accuracy to our full feature set, implying that we may have overlap in our full set. However, our test data in this experiment only included 3 behaviours; our state features resulted from analyzing 19 different behaviours types, and so it's possible that our full feature set may have higher accuracy over a larger variety of behaviours, though further testing would be required to validate this. Completely new features are also a likely source of improvement. For instance, our current feature set cannot learn behaviours that require a change in speed, limiting the behaviours we can generate. Other features may not enable new behaviours, but improve current accuracies. For example, adding a feature that calculates distance in terms of shortest possible path (taking into account walls, etc.) as opposed to Euclidean distance may improve accuracies in more complicated environments.

Our definition of a generated behaviour's accuracy should be taken into account when interpreting our results. PaintBoard's goal was to *prototype* behaviours, yet our definition of a successful prediction is very strict (an exact match between corresponding grid cells), and as such, lower accuracies may be permissible. For example, a snapshot that performs well in practice may score poorly with our metric if the colours were offset by one square either vertically or horizontally. Additional accuracy metrics may be more appropriate, such as counting a match if the desired colour is in a nearby cell. Finally, as our input storyboard provides very limited data for relatively complex behaviours, it is highly unlikely to achieve near-100% accuracies with generic techniques.

Modern games are commonly not discrete grid-like areas, and have characters moving about smoothly in a continuous space. In order to bring PaintBoard to such spaces in its current form, many algorithmic problems must be solved. One problem in doing this is that calculating state features for every pixel in the game world is computationally too expensive in most cases. A naïve solution could be range thresholds, perhaps definable by the author, that limit how far away from the user and computer characters the calculations take place. A solution to combat the complexity of a continuous space could be to discretize the space—placing an artificial grid, similar to what we already use in PaintBoard, over the continuous area. This algorithmic trick could even be invisible to the author: allow them to paint in a continuous space, and apply the grid only when calculating the state features.

6.5. Conclusion

The PaintBoard algorithm was sufficient for our initial exploration of our painting-storyboards approach, and enabled rapid, iterative prototyping of interactive behaviours. Our initial choice of radial basis function SVM and state feature set used in our workshop proved to be adequate choices when compared to other learning techniques used with our PaintBoard system. However, different machine learning algorithms did not yield performance gains, suggesting that a fundamentally different approach for behaviour generation may be necessary for better accuracy. Further, our state features used to train our SVM were able to successfully reproduce behaviours to a reasonable (sufficient for prototyping) level, as indicated by our performance measurements. However, our analysis suggested that our initial set may contain redundant features, and is likely incomplete, therefore a more systematic investigation to identify possible missing features is important future work.

7. Conclusion

This work detailed a novel technique for prototyping interactive character behaviours by painting and storyboarding. We presented results from exploratory interviews and a programmer study, which informed our development and interface design, resulting in PaintBoard: a novel painting and storyboarding interface. The algorithm we developed is based on machine learning and can generate real-time interactive behaviours based solely on a few digitally painted snapshot examples. As part of this, we developed a novel feature set (state features) that can represent important characteristics of paired interactive behaviours which may be useful to other algorithms in the same problem domain. Further, we conducted a workshop where people used PaintBoard, and the results highlighted the strengths of our approach, showing how developers, with minimal training, can easily use PaintBoard to prototype behaviours. Finally, we devised a metric to evaluate the quality of generated behaviours and used it to explore variants of our algorithmic approach. This lead to us discovering that little can be gained by simple algorithm or parameter adjustment in our system and current algorithmic approach.

7.1. Limitations and Future Work

This work served as a proof-of-concept of PaintBoard's approach of enabling people to generate behaviours by painting storyboards. However, the interface, algorithm, and evaluation methodology all require further study to make behaviour generation authoring more fluid, accessible, and reliable. In spite of these limitations, we have shown support for PaintBoard's approach, and it extends previous work to provide a baseline for future work in authoring interactive behaviours.

7.1.1. Authoring Interface and Interaction

Beyond the additions mentioned in our workshop, such as branching storyboards and painting with different weights, we need to explore extending the painting metaphor to enable more interactive behaviours. For example, it is not immediately obvious how to paint non-static properties such as character movement speed. Another challenge is the inherent chronology of storyboards. PaintBoard should be extended to consider the order of the snapshots in the storyboard, as people expect them to be chronological; currently, this is not handled in the interface, nor the algorithm, and may be useful in learning more complicated behaviours. Temporally constraining each snapshot, however, further constrains an already limited input set (a short storyboard), and requiring additional data from a user may limit their productivity with PaintBoard. This remains challenging and important future work.

Looking to successful painting software for inspiration, it may be useful to explore how PaintBoard could use layers (e.g., as used in Adobe Photoshop[5]or the GIMP[6] software packages). This enables users to separate varying aspects of what they are drawing, and may be useful for representing speed, or other features such as character orientation, without cluttering the snapshots. For example, temporal behaviour aspects, such as speed of movement, might be expressed on its own layer, possibly with interactions that benefit those parts of the behaviour.

7.1.2. Behaviour Generation and State Features

We discussed many of PaintBoard's algorithmic limitations, such as our definition of a correct prediction, in Section 6.4, but we have, to our knowledge, been the first to attempt to quantitatively evaluate generated interactive behaviours. This provides a solid baseline for future work including even entirely different authoring techniques that change the way users create behaviours, the data to be analyzed, or the generation technique itself.

While the subset of algorithms we compared in our study was diverse, it was incomplete. However, our results suggested that general machine learning techniques will not easily provide significant performance increases. Thus, we suggest that new machine learning or analytical techniques tailored specifically for learning interactive behaviours may be a promising direction.

Different approaches to state features may greatly improve the accuracy of generating behaviours as well as enable an even wider variety of behaviour that can be authored. While a subset of our

[5] http://www.photoshop.com/

[6] http://www.gimp.org/

features seemed to perform as well as the full set, we caution that our evaluation was only tested with a small number of behaviours, and our accuracy metric was strict and possibly not suited to evaluate prototyping environments. Our results simply highlight that selection of features for analyzing storyboard input is non-trivial, and that different feature sets may have similar performance for specific subsets of behaviours. Fundamentally different approaches to state features may enable more complicated behaviours; for example, instead of calculating state features at the grid cell level, higher level features that encapsulate state information (e.g. the user character is moving towards the treasure) may be enable behaviours that rely upon knowing past movements.

7.1.3. Further Evaluations

Our studies were high-level and focused on obtaining a general sense of PaintBoard's qualities, but targeted follow up studies with more rigorous evaluations need to be conducted; we could perform a follow-up study targeted at PaintBoard's potential as a communications tool where developers and non-technical designers work together to create a behaviour, and use PaintBoard as the prototyping and communication medium. This will enable us to more concretely reflect on PaintBoard's potential as a communication and prototyping tool in a real-world scenario. As previously mentioned, new evaluations targeted at improving PaintBoard's algorithm or state features should look to refine our method of measuring the accuracy of a generated behaviours as well. For example, being less strict on what counts as a successfully painted cell (as PaintBoard is a prototyping tool), or even evaluating the behaviour on the scene level instead of the grid level may be an improvement.

7.2. Contributions

Contributions of our research include:

1. A novel interaction method—digitally painting storyboards—that enables the description and real-time testing of interactive behaviours for computer controlled characters.

2. An original interface that enables 1.

3. A behaviour generation algorithm that can quickly generate a real-time interactive behaviour from a painted storyboard.

4. A workshop with developers and interviews with professionals in the game industry that grounded our design and development of 1, 2, and 3, and provide a baseline understanding of some approaches of creating interactive behaviours.

5. An evaluation of PaintBoard's overall approach of painting storyboards

6. An evaluation of PaintBoard's behaviour generation algorithm.

Overall, PaintBoard serves as a proof of concept for how real-time interactive behaviours can be prototyped through painting and storyboarding, and provides an initial solution to the interface and algorithmic design problems. Our interface and interaction design was evaluated with a design workshop, and our algorithmic solutions were analyzed by comparing different learning algorithms with PaintBoard. The evaluations highlighted the success of the painting-storyboards approach, and we envision that PaintBoard will serve as a proof of concept and baseline for future work, for example, extending PaintBoard into 3D continuous worlds. We hope that tools for

simplifying the authoring of interactive behaviours will improve, making interactive content creation faster and more accessible to a broader audience

8. Bibliography

1. Abeel, T., de Peer, Y. V., and Saeys, Y. Java-ML: A Machine Learning Library. *Journal of Machine Learning Research 10*, (2009), 931–934.

2. Berg, B.L. Open Coding. In *Qualitative Research Methods for the Social Sciences*. 1989, 364 – 366.

3. Beyer, H. and Holtzblatt, K. *Contextual design: defining customer-centered systems*. Elsevier, 1997.

4. Breiman, L. Random Forests. *Machine learning 45*, 1 (2001), 5–32.

5. Buxton, B. *Sketching User Experiences: Getting the Design Right and the Right Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

6. Caruana, R. and Niculescu-Mizil, A. An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, ACM Press (2006), 161–168.

7. Chang, C.-C. and Lin, C.-J. LIBSVM. *ACM Transactions on Intelligent Systems and Technology 2*, 3 (2011), 1–27.

8. Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

9. Diethelm, I. Systematic story driven modeling: a case study. *"Third International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM04)" W5S Workshop - 26th International Conference on Software Engineering*, IEE (2004), 65–70.

10. Dix, A., Finlay, J.E., Abowd, G.D., and Beale, R. *Human-Computer Interaction.* Prentice Hall PTR, 2003.

11. Dontcheva, M., Yngve, G., and Popović, Z. Layered acting for character animation. *ACM SIGGRAPH 2003 Papers on - SIGGRAPH '03*, (2003), 409.

12. Förger, K., Takala, T., and Pugliese, R. Authoring Rules for Bodily Interaction: From Example Clips to Continuous Motions. *Intelligent Virtual Agents*, (2012), 341–354.

13. Forte, D., Gams, A., Morimoto, J., and Ude, A. On-line motion synthesis and adaptation using a trajectory database. *Robotics and Autonomous Systems 60*, 10 (2012), 1327–1339.

14. Garcia, E.A. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering 21*, 9 (2009), 1263–1284.

15. Gebhard, P., Kipp, M., Klesen, M., and Rist, T. Authoring scenes for adaptive, interactive performances. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems - AAMAS '03*, (2003), 725.

16. Gleeson, B., Maclean, K., Haddadi, A., Croft, E., and Alcazar, J. Gestures for industry: intuitive human-robot communication from human observation. *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction (HRI '13)*, IEEE Press (2013), 349–356.

17. Goldman, D.B., Curless, B., Salesin, D., and Seitz, S.M. Schematic storyboarding for video visualization and editing. *ACM Transactions on Graphics 25*, 3 (2006), 862.

18. Greenberg, S., Carpendale, S., Marquardt, N., and Buxton, B. The narrative storyboard: telling a story about use and context over time. *interactions 19*, 1 (2012), 64–69.

19. Hernandez, F.E. and Francisco, R., 2011. Reducing Video Game Creation Effort with Eberos GML2D. In M. Ansari, ed., *Game Development Tools*. CRC Press, 2011, 267.

20. Igarashi, T. and Hughes, J.F. Smooth meshes for sketch-based freeform modeling. *ACM SIGGRAPH 2007 courses on - SIGGRAPH '07*, ACM Press (2007), 22.

21. Igarashi, T., Matsuoka, S., and Tanaka, H. Teddy: a sketching interface for 3D freeform design. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, ACM Press (1999), 409–416.

22. Igarashi, T., Moscovich, T., and Hughes, J.F. Spatial keyframing for performance-driven animation. *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05*, ACM Press (2005), 107.

23. Igarashi, T., Moscovich, T., and Hughes, J.F. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics 24*, 3 (2005), 1134.

24. Ju, E., Choi, M.G., Park, M., Lee, J., Lee, K.H., and Takahashi, S. Morphable crowds. *ACM Transactions on Graphics 29*, 6 (2010), 1.

25. Kelly, S. and Tolvanen, J.-P. *Domain-specific Modeling: Enabling Full Code Generation.* Wiley, 2008.

26. Kohavi, R. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, (1995), 1137–1143.

27. Landay, J.A. and Myers, B.A. Interactive sketching for the early stages of user interface design. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '95*, ACM Press (1995), 43–50.

28. McNaughton, M; Cutumisu, M; Szafron, D; Schaeffer, J; Redford, J; Parker, D. ScriptEase : Generative Design Patterns for Computer Role-Playing Games. *Proceedings of the 19th IEEE international conference on Automated software engineering*, IEEE Computer Society Washington, DC, USA (2004), 386–387.

29. Peszko, J.P. 'Assassin's Creed': Redefining the Action Game. 2007. http://www.awn.com/vfxworld/assassins-creed-redefining-action-game.

30. Pizzi, D. and Cavazza, M. From Debugging to Authoring : Adapting Productivity Tools to Narrative Content Description. *Lecture Notes in Computer Science 5334*, (2008), 285–296.

31. Pugliese, R. and Lehtonen, K. A Framework for Motion Based Bodily Enaction with Virtual Characters. *Intelligent Virtual Agents*, (2011), 162–168.

32. Reynolds, C.W. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics 21*, 4 (1987), 25–34.

33. Shen, E.Y. and Chen, B. Toward gesture-based behavior authoring. *International 2005 Computer Graphics*, (2005), 59–65.

34. Shneiderman, B. Creativity support tools: accelerating discovery and innovation. *Communications of the ACM 50*, 12 (2007), 20–32.

35. Suay, H.B., Toris, R., and Chernova, S. A Practical Comparison of Three Robot Learning from Demonstration Algorithm. *International Journal of Social Robotics 4*, 4 (2012), 319–330.

36. Terry, M. and Mynatt, E.D. Recognizing creative needs in user interface design. *Proceedings of the fourth conference on Creativity & cognition - C&C '02*, (2002), 38–44.

37. Ulicny, B., Ciechomski, P. de H., and Thalmann, D. Crowdbrush. *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '04*, ACM Press (2004), 243.

38. Walter, R. and Masuch, M. How to integrate domain-specific languages into the game development process. *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology - ACE '11*, ACM Press (2011), 1.

39. Wengraf, T. *Qualitative research interviewing: Biographic narrative and semi-structured methods.* Sage, 2001.

40. Wolber, D. Pavlov: Programming by stimulus-response demonstration. *Proceedings of the SIGCHI conference on Human …*, (1996), 252–259.

41. Young, J.E., Sharlin, E., and Igarashi, T. Teaching Robots Style: Designing and Evaluating Style-by-Demonstration for Interactive Robotic Locomotion. *Human–Computer Interaction 28*, 5 (2013), 379–416.

42. Yu, K., Wang, H., Liu, C., and Niu, J. Interactive Storyboard : Animated Story Creation on Touch Interfaces. *Active Media Technology*, (2009), 93–103.

Appendix A: Materials Used in our Initial Investigations

- Ethics approval certificate
- Recruitment e-mail
- Informed consent form
- Semi-structured interview outline

UNIVERSITY OF MANITOBA | Research Ethics and Compliance
Office of the Vice-President (Research and International)

Human Ethics
208-194 Dafoe Road
Winnipeg, MB
Canada R3T 2N2
Phone +204-474-7122
Fax +204-269-7173

APPROVAL CERTIFICATE

May 24, 2013

NSERC, URGP

TO: **James E. Young**
Daniel Rea
Principal Investigators

FROM: **Susan Frohlick, Chair**
Joint-Faculty Research Ethics Board (JFREB)

Re: **Protocol #J2013:070**
"User Generated Interactive Behaviour Creation for Automated Agents"

Please be advised that your above-referenced protocol has received human ethics approval by the **Joint-Faculty Research Ethics Board**, which is organized and operates according to the Tri-Council Policy Statement (2). **This approval is valid for one year only.**

Any significant changes of the protocol and/or informed consent form should be reported to the Human Ethics Secretariat in advance of implementation of such changes.

Please note:

- **If you have funds pending human ethics approval, please mail/e-mail/fax (261-0325) a copy of this Approval (identifying the related UM Project Number) to the Research Grants Officer in ORS in order to initiate fund setup. (How to find your UM Project Number: http://umanitoba.ca/research/ors/mrt-faq.html#pr0)**

- **if you have received multi-year funding for this research, responsibility lies with you to apply for and obtain Renewal Approval at the expiry of the initial one-year approval; otherwise the account will be locked.**

The Research Quality Management Office may request to review research documentation from this project to demonstrate compliance with this approved protocol and the University of Manitoba *Ethics of Research Involving Humans.*

The Research Ethics Board requests a final report for your study (available at: http://umanitoba.ca/research/orec/ethics/human_ethics_REB_forms_guidelines.html) **in order to be in compliance with Tri-Council Guidelines.**

Dear <name>,

I am very sorry to bother you. My name is Daniel Rea, and I am a Masters student at the University of Manitoba working on tools to assist game design and development. I am looking for people who work professionally in the gaming industry who I can interview about their experience of designing games, especially in regard to the creation of Non-Player Characters (NPCs). We are looking at developing new methods or interfaces to assist with quick prototyping of NPC behaviours.

Participation in this study will take approximately 30-45 minutes of your time.
You will receive an Amazon gift certificate for $10. All information you provide is considered completely confidential; your name will not be included or in any other way associated with the data collected in the study. The interview will be done over Skype, and will be recorded for analysis: however, no data (e.g., audio clips) will be shared or disseminated, and any quotes or verbal statements will be completely anonymized if included in research publications. This study was reviewed and approved by the University of Manitoba's Ethics Board

If you are interested, please reply to this email for more information or to set up an interview time.

In addition, we are still looking for more participants so if you have any colleagues or friends who have professionally developed games and might be interested in this study, we would greatly appreciate it if you forwarded this to them.

Sincerely,

Daniel Rea,

MSc. Student, Human-Computer Interaction Lab,

University of Manitoba

Consent form to be read verbally to participants (for the interview)
Research Project Title: User Generated Interactive Behaviour Creation for Automated Agents
Researcher: Dr. James Young [redacted], (young@cs.umanitoba.ca), Daniel Rea (daniel.rea@cs.umanitoba.ca)

This consent form is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, you should feel free to ask.

You are invited to complete an interview on your experience of designing games, especially in regard to the creation of Non-Player Characters (NPCs). We are hoping to create a new interface and algorithm that will allow quick prototyping of behaviours without the need for explicit programming. These questions will concern your history in game design. After the interview, feel free to tell us if you have any additional feedback on the research project or ask any other questions you might have.

Participation in this study is voluntary, and will take approximately 30-45 minutes of your time.
You will receive an Amazon gift certificate for $10. This interview will be taped, however, all information you provide is considered completely confidential; your name will not be included or in any other way associated, with the data collected in the study. The recordings will not be released and will be encrypted and stored in a secure location. Additional information that might be identifying such as project titles and company names will also be omitted. Data collected during this study will be retained until publication in a locked office in the EITC building, University of Manitoba, to which only researchers associated with this study have access. However, the University of Manitoba may look at research records to see that the research is being done in a safe and proper way." Once published, results of the study will be made available to the public for free at http://home.cs.umanitoba.ca/~young/. There are no known or anticipated risks associated to participation in this study.

Your verbal consent indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a subject. By doing this you also confirm that you of the age of majority in Canada (18 years or more). In no way does this waive your legal rights nor release the researchers, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from the study at any time, and /or refrain from answering any questions you prefer to omit, without prejudice or consequence. Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation.

This research has been approved by the Joint-Faculty Research Ethics Board**.** If you have any concerns or complaints about this project you may contact Dr. James Young at [redacted] or the Human Ethics Secretariat at [redacted]. An electronic copy of this consent form has been provided to you.

Do you agree to participate in this study?

Interview Outline

main goal: to identify what NPC behaviours game designers want to create for their games

why? Obviously we want to show people we can create behaviours people want

side goal: do they have things they like or dislike about their current behaviour design workflow? What is that workflow?

why? Can be used to justify or inform our UI design choices.

Below follows a rough flow of the interview...

Consent----------

Read the consent form. If they do not agree, the interview does not occur.

Intro----------

we want to investigate:

creating NPCs whose behaviours

use the environment

react dynamically to the player

and the creation process of such behaviours

Emphasize that interaction with the player is key here.

Questions----------

Misc Demographics----------

(audience, accessibility)

How long have you been working on games?

Did you receive formal programming experience? Learnt on the job?

Would you consider yourself a professional or indie developer?

Would you mind telling me what company you work for?

Kinds of behaviours---------- (for the list of behaviours)

What kind of games have you worked on?

Within these games, what kind of NPCs were there? (High level, like Bard, Soldier, etc)

What did these NPCs do in the game?

What were some typical physical interactions the NPC had with the player? (actions)

for example,

RTS: troop movement

RPG: townspeople moving, guards guarding, thieves sneaking, bards performing, beggars begging, hawkers hawking

FPS: combat, formations, "AI"

Are there certain behaviours that are difficult to create because of interactions with the player?

Did these actions have to consider the environment? How so?

Creation---------- (about workflow, authoring process)

What do you use to create these? Programming?

How did you create these behaviours (from the above conversation)?

i.e. algorithms, machine learning, etc

development time of creating NPC behaviours? (Roughly...long? Short?)

What do you spend the most time on while making the behaviours?

have you considered other means: libraries? AI? simulation packages? "roll your own"? others?

What do you think is the biggest hindrance in general while trying to program these NPCs?

Do you find picturing what's going to happen from your code easy? Is this a problem for NPC behaviour creation?

Results----------

satisfaction behaviours -- have you been satisfied with the behaviours you've created?

If not, why? Why could you not improve them?

Armchair Design----------

(especially if they have a lack of experience in 1st hand creation)

Are there interactive NPC behaviours you've seen in games and liked a lot?

Some you've thought are particularly bad? Why?

Are there types of NPCs you'd like to try to create?

Appendix B: Materials used in our Workshop and Algorithm Evaluation

- Ethics amendment certificate
- Recruitment e-mail
- Informed consent form
- Post-Workshop Questionnaire

UNIVERSITY OF MANITOBA | Research Ethics and Compliance
Office of the Vice-President (Research and International)

Human Ethics
208-194 Dafoe Road
Winnipeg, MB
Canada R3T 2N2
Phone +204-474-7122
Fax +204-269-7173

AMENDMENT APPROVAL

September 25, 2013

TO: **James E. Young**
Daniel Rea
Principal Investigators

FROM: **Susan Frohlick, Chair** [redacted]
Joint-Faculty Research Ethics Board (JFREB)

Re: **Protocol #J2013:070**
"User Generated Interactive Behaviour Creation for automated Agents"

This will acknowledge your request dated August 29, 2013, but received September 16, 2013, requesting amendment to your above-noted protocol.

Approval is given for this amendment. Any further changes to the protocol must be reported to the Human Ethics Secretariat in advance of implementation.

Dear <name>,

I am very sorry to bother you. My name is Daniel Rea, and I am a Masters student at the University of Manitoba working on tools to assist game design and development. I am looking for people who work professionally in the gaming industry who are interested in testing our new prototype software for designing Non-Player Character (NPC) movement styles without any coding. We are looking for feedback to steer future research directions, as well as justification for or against our approach.

Participation in this study is voluntary, and will take approximately 90 minutes of your time. It will consist of a 15 minute tutorial, at most 60 minutes of free creation time where you are asked to experiment with our system, and 15 minutes or less of filling in a questionnaire. You will receive $15 cash in compensation. The session will be video recorded, however, all information you provide is considered completely confidential; your name will not be included or in any other way associated, with the data collected in the study. The recordings and collected data will be used for analysis only, and video will only be used for dissemination only with your express consent. Any quotes or verbal statements will be completely anonymized if included in research publications. This study was reviewed and approved by the University of Manitoba's Ethics Board.

If you are interested, please reply to this email to book a spot for the work shop. The tentative date is <date dependent on ethics approval>. It is appreciated if you bring your own laptop with the (free) Java Runtime Environment installed (www.java.com). However, computers can be provided for you, so please let us know in your email.

In addition, we are still looking for more participants so if you have any colleagues or friends who have professionally developed games and might be interested in this study, we would greatly appreciate it if you forwarded this to them.

Sincerely,

Daniel Rea,

MSc. Student, Human-Computer Interaction Lab,

University of Manitoba

Research Project Title: User Generated Interactive Behaviour Creation for Automated Agents
Researcher: Dr. James Young [redacted], (young@cs.umanitoba.ca), Daniel Rea (daniel.rea@cs.umanitoba.ca)

This consent form is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, you should feel free to ask.

You are invited use your experience of designing games to help us test a prototype tool used to create Non-Player Character (NPC) movement styles. We use a new painting interface and an algorithm that allows quick prototyping of behaviours without the need for explicit programming. After you use our software, we will save the behaviours you have created for viewing later, and ask you to fill in a questionnaire about your thoughts about the pros and cons of our prototype. At any time, feel free to tell us if you have any additional feedback on the research project or ask any other questions you might have.

Participation in this study is voluntary, and will take approximately 90 minutes of your time.
You will receive $15 cash in compensation. The session will be video recorded, however, all information you provide is considered completely confidential; your name will not be included or in any other way associated, with the data collected in the study. The recordings and collected data will be used in our analysis but will only be used in dissemination with your permission and be encrypted and stored in a secure location. Data collected during this study will be retained for a period of five years (to allow follow-up analysis) in a locked office in the EITC building at the University of Manitoba, to which only researchers associated with this study have access. However, the University of Manitoba may look at research records to see that the research is being done in a safe and proper way. Once published, results of the study will be made available to the public for free at http://home.cs.umanitoba.ca/~young/. There are no known or anticipated risks associated to participation in this study.

For purposes of research analysis the experiment will be videotaped. By signing this consent form, you agree that you understand this and that we may use the video for data analysis purposes.

Your signed consent indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a subject. By doing this you also confirm that you of the age of majority in Canada (18 years or more). In no way does this waive your legal rights nor release the researchers, sponsors, or involved institutions from their legal and professional responsibilities. At any time, you are free to withdraw from the study, and /or refrain from answering any questions you prefer to omit and /or revoke your consent without giving any reason for doing so and without any disadvantage or reprisal for withdrawal or refusal. Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation.

This research has been approved by the Joint-Faculty Research Ethics Board**.** If you have any concerns or complaints about this project you may contact Dr. James Young at [redacted]1 or the Human Ethics Secretariat at [redacted]. An electronic copy of this consent form has been provided to you.

Participant's Signature _____ Date _____

Researcher's Signature _____ Date _____

Post-study questionnaire

1. Can you describe your overall experience of using our tool for creating interactive behaviours?

2. List some good things you noticed about our overall method?

3. List some shortfalls of our method?

4. Were there any behaviours that were easy to create? What was easy about it?

5. Were there any behaviours that were difficult to create? What was difficult about it?

6. Do you see this tool as being useful at any point in your workflow? If so, how would you use it? If not, why?

7. Any additional ideas for improvement?

8. Final comments or suggestions?

CONSENT TO USE YOUR WORK IN DISSEMINATION OF THIS RESEARCH

I hereby give my permission to use:

- ☐ The behaviours I created
- ☐ My questionnaire answers
- ☐ Recorded video that I am visible in
  - ☐ My face must be blurred so it cannot be recognized as me
- ☐ Recorded video that I am audible in
  - ☐ My voice must be muffled so it cannot be recognized as my voice

in the dissemination of this research. I understand that I have 2 weeks from the end of this experiment in which I can change this decision and can do so by sending an e-mail to [redacted] detailing which permissions I wish to give or withdraw.

Date: _____

Signature: _____