COMPLEMENTARY COMPANION BEHAVIOR IN VIDEO GAMES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Gavin Scott

June 2017

COMMITTEE MEMBERSHIP

TITLE:	Complementary Companion Behavior in Video Games
AUTHOR:	Gavin Scott
DATE SUBMITTED:	June 2017
COMMITTEE CHAIR:	Foaad Khosmood, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Alexander Dekhtyar, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Franz Kurfess, Ph.D. Professor of Computer Science

ABSTRACT

Complementary Companion Behavior in Video Games

Gavin Scott

Companion characters in are present in many video games across genres, serving the role of the player's partner. Their goal is to support the player's strategy and to immerse the player by providing a believable companion. These companions often only perform rigidly scripted actions and fail to adapt to an individual player's play-style, detracting from their usefulness. Behavior like this can also become frustrating to the player if the companions become more of a hindrance than they are a benefit. Other work, including this project's precursor, focused on building companions that mimic the player. These strategies customize the companion's actions to each player, but are limited. In the same context, an ideal companion would help further the player's strategy by finding complementary actions rather than blind emulation.

We propose a game-development framework that adds complementary (rather than mimicking) companions to a video game. For the purposes of this framework a "complementary" action is defined as any that furthers the player's strategy both in the immediate future as well as in the long-term. This is determined through a combination of both player-action and game-state prediction processes, while allowing the companion to experiment with actions the player hasn't tried. We used a new method to determine the location of companion actions based on a dynamic set of regions customized to the individual player. A user study of game-development students showed promising results, with a seventeen out of twenty-five participants reacting positively to the companion behavior, and nineteen saying that they would consider using the framework in future games.

ACKNOWLEDGMENTS

Thanks to Dr. Foaad Khosmood for his advice and assistance throughout the development of this project. Further thanks to Dr. Franz Kurfess and Dr. Alexander Dekhtyar for taking the time to review this thesis, and to Daniel Toy, who worked in parallel on a related project and often offered assistance. Finally, I would like to thank everyone that participated in the user study and everyone (listed in the appendix) whose art we used in our example games.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Description of the Problem

As video games evolve they become more and more complex; AAA games have grown from simple, two-dimensional games to complex, fully-explorable worlds with rich stories and hundreds of hours of unique game-play. With this complexity the need to integrate artificial intelligence (AI) into games has grown [40], in some cases to adapt the game mechanics to the style of a particular player [27], and in others to make the characters in the game more realistic. For a player to immerse themselves in a game's environment it needs to be believable, which often requires a vast amount of computer-controlled "non-player characters" (NPCs) that must interact with both each other and the player in intuitive ways [45].

These NPCs can be broadly divided into two categories: opponents or companions. Other NPCs include passive characters like shopkeepers or villagers, but they generally don't require the same level of intelligence because the level of player interaction with them is much lower. Much of the research related to video game AI has been done to create believable and challenging opponents for the player. To keep the player engaged in the game the developers strive to make their opponents as smart as possible, preventing the game from becoming too easy or repetitive. As a result, games are regularly published that have enemies with unpredictable, complex strategies or with nuanced goals and hidden agendas that keep the player engaged with the game every time they play [24].

It is more difficult to find research focused on creating believable — and most importantly, useful — companion AI. The purpose of a companion in a video game is

to assist the player in whatever they are trying to accomplish, in some cases finding complementary actions that help the player achieve their goal more easily than they would have on their own. This is rarely attempted in most games, and the companion is often relegated to a beast of burden or a lackey that follows and mimics the player at every turn [23]. Unrealistic behavior like this can interfere with the game's immersion. In the worse case, a poorly-done companion can seriously worsen the overall game experience and frustrate the player by constantly getting in their way.

A teammate should work with the player, changing their strategy to suit the player and in the process make the game more interesting, engaging, and enjoyable. Following a small, scripted set of decisions is not enough to promote this engagement, and it can be detrimental to the quality of the game as a whole. Scripted behavior only covers a finite set of situations, making it possible for situations to arise where the character does not know how best to act. By using AI to dynamically make decisions this can be avoided, resulting in more believable characters and a more immersive game.

1.2 Overview of the Solution

This project extends the work started by Travis Angevine, who began a game development framework that added an AI companion which learned from and mimicked the player [15]. We modified this framework to further decouple it from its example game, as well as to improve the effectiveness of the companion by looking for actions that complement the players strategy rather than copy it. We defined a complementary action as any action (not necessarily the same as the player's action at the time of the decision) that is beneficial to the player's strategy. Determining to what degree one action is complementary to a strategy is very hard to quantify, so we used human players to gauge the companion's success.

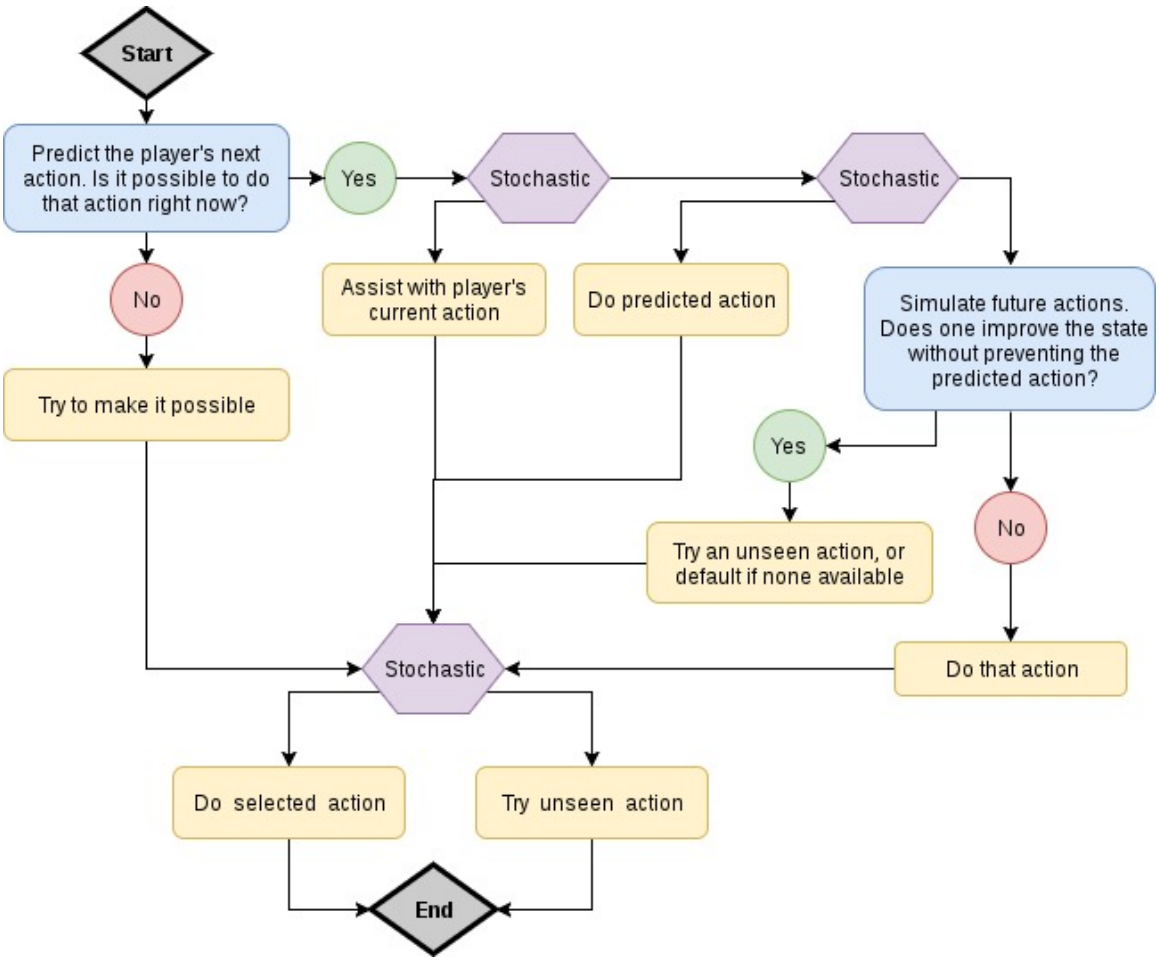

Figure 1.1: A simplified flowchart showing an overview of the complementary decision-making process

Specifically, we developed a complementary decision making process that combines evaluating the player's current and past actions, predicting their likely next action, predicting future game states through simulation, and experimenting with unseen actions to increase the companion's range of possibilities. This allows the actions the companions take to be tailored to the behavior of the current player without any need for scripted behaviors or previous training. A simplified version of this process is shown in figure 1.1. We also implemented a new dynamic region system for determining where actions should take place that balances putting the decision in the hands of the framework while allowing the game designer the freedom to make fine-tuned, game-specific location decisions.

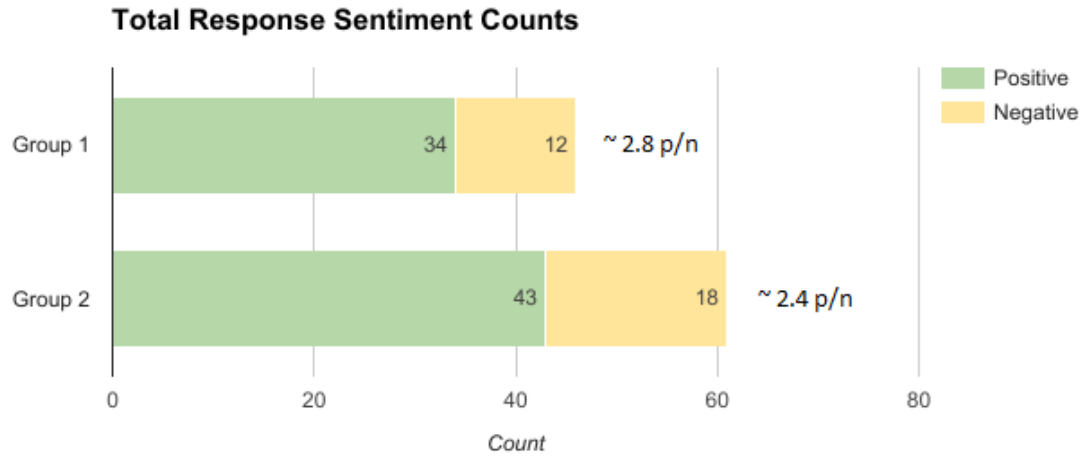

Figure 1.2: A summary of companion-focused response sentiments across all questions, split by group. The numbers at the ends of the bars represent the ratio of positive to negative comments in that group.

To evaluate the system a user study was conducted; twenty-five game-development students split into two groups were asked to play a game using the framework, providing feedback through a survey. Survey questions focused primarily on the general strengths and weaknesses of the game, the behavior of the companion, and how useful the framework would be as a development tool. The results were promising; seventeen participants reacted positively to the companion's behavior, and nineteen indicated that they would consider using the framework in their future games. Two users even mentioned, before being asked specifically about the companion or knowing the purpose of the project, that the companions actions matched what they would expect from playing the game with another human.

As a basic overview of the results, figure 1.2 shows the number of comments about the companion across all questions, broken down by sentiment and split into the two groups of students. Each group had at least twice as many positive comments than negative, and we expect this ratio may be able to be increased further if some technical limitations are addressed, discussed later in the paper.

1.3 Outline of the Paper

Chapters 2 and 3 contain background information on the different components of this project as well as related work. Chapter 4 describes the complementary decision making process that the companion uses, and chapters 5 and 6 go into how that process was implemented. Chapter 7 describes the user study we used to evaluate the final product and chapter 8 analyzes its results. Finally, chapter 9 concludes with a summary of contribution and some of the challenges faced during development, and chapter 10 describes potential future work.

Chapter 2

BACKGROUND

This section provides an overview of some of the background research related to video games and game AI. It discusses intelligent agents, how learn-by-observation methods can be used to implement them, and how AI is used in video games to dynamically change gameplay and create interesting companions. It also describes the classifiers included in this framework, as well as some background information on Unity, the game engine it's built in.

2.1 Learn by Observation

This framework is based on "Learn by Observation" (LBO) techniques. LBO, also referred to as "Learn by Demonstration," is a method of supervised learning where an intelligent agent learns how to behave by watching another agent. The agent being watched is often a human, and in this case is the player while the observer is the companion. In general, data is recorded as a human performs a task and after some amount of time the agent begins performing the task as well. Recorded data will contain a set of variables representing some state, as well as the action the human took. Traditionally, when the agent is making a decision it evaluates the current state and tries to identify which action the human would take based on the state-action pairs in the recorded history [33, 32, 34].

A mimicking companion in a video game would use this technique to determine which action to perform. However, because this framework's purpose it to create a complementary companion rather than a mimicking one, this technique is used differently. Rather than recording unordered sets of vector-action pairs, the framework

orders the data and learns how to predict the player's next action instead of what they would do now. Their predicted future action is used in the decision-making process instead of being returned as the companion's decision, but the underlying learn-by-observation process is the same.

2.2 Adaptive Gameplay & A.I.

Adaptive gameplay is a game-development technique that attempts to dynamically change elements of the game based on the behavior of the current player [29], which is often used to make the game more engaging and prevent player frustration. An example of this technique is through "negative feedback" [38]. This process observes the players progress, noting how quickly they are advancing or how often something "bad" happens to them. If they are doing too well the difficulty is increased until their progress falls below a level fixed by the developer. Similarly, if they are not doing well enough the difficulty is decreased. This keeps the player from becoming too bored or frustrated, and also gives the developer some more control over the pace of the game.

The AI in a game can also be adapted to each player, further increasing the quality of the gameplay. Adaptive game AI can be applied to a player's teammates [13, 31, 43], but is also often applied to the enemy AI [18]. In either case, the computer-controlled character learn to react to each individual player differently; applied to teammates this makes the relationship between the player and the team more useful and rewarding, and applying it to enemies makes the game more challenging and immersive. This project applied these techniques to the companion AI, to develop a teammate that is more effective than one with statically-defined behavior.

2.3 Classifiers

This section overviews the supervised-learning classifiers that were implemented for this framework. Decision Trees and Naive Bayes are potential methods for choosing an action-location pair from the player's data based on the current game state, and Linear Regression is a potential alternative for predicting future game states. Only one classifier, specified by the developer, is used at a time and a generic class is included in the framework for a developer to extend if a different classifier is required. All of the classifiers rely on player data to train them before they are used, and can be retrained at any point during the gameplay to become updated with any new data that has been gathered since they were last trained. In our example games, we used the Decision Tree classifier.

2.3.1 Decision Trees

Decision trees, which are used in our example games, classify an input by traversing a tree where each node represents a decision. The tree must be built beforehand based on example data with matched input-output pairs. This process evaluates each input feature vector, adding nodes as necessary such that traversing the tree with the provided input vectors results in as few misclassifications as possible. Each node represents a specific check on the data, evaluating to true or false based on a specific value in the input. One of two branches is taken based on the outcome of each check, and the next node in the tree determines the next decision. Leaf nodes at the ends of branches represent classes, and are returned when the traversal reaches a leaf during the classification process [36].

2.3.2 Naive Bayes

Naive Bayes classifiers use the statistical probabilities found in the training data to perform classifications. Therefore, like Decision Trees, Naive Bayes classifiers also require training with example data before they can be used. By evaluating both the probability of each outcome class as well as the probability for each class given each value present in the input vector the classifier is able to calculate which outcome is the most likely. The most likely outcome based on these probabilities is returned as the result. It should be noted that the multiplication of so many probabilities can result in too small of a number; this is solved by summing the natural logs of the probabilities rather than multiplying them. This ensures that the most likely outcomes are returned while keeping the probabilities more manageable [37].

2.3.3 Linear Regression

Linear Regression is a technique to map a vector of numeric features to a numeric class. Essentially, this process is done by examining example data and identifying the best equation for a line that maps the input features (independent variables) to the correct value. The equation for the line is found that most closely matches the training data, minimizing the error between the output and the expected value. After training, new input vectors can quickly be plugged in to this equation to find a predicted value [42].

This framework's Linear regression implementation was initially intended for predicting the state of the game by creating a separate classifier for each unique feature in the input vectors, using the other features to predict it. However, due to the nature of the data as well as its limited quantity (it must be gathered from each player before a companion can enter the game), it was replaced by the simulation process detailed in section 6.4. The simulation process was also favored over linear regression

because it doesn't limit the game state features to numerical data, allowing the game developer more freedom.

2.4 Tools

MimicA, the framework upon which this project was based, was built using the Unity game engine [15, 11]. Unity is one of the more popular game engines on the market, both for 2D and 3D games, so writing a framework that is compatible with it allows a large audience to access it. It handles many of the tasks common to all games, like creating "scenes," populating them with objects (like the player character), and managing sprites and animations. It should be noted that Unity supports multi-threading through its "Coroutines," but the management of game objects is limited to the main thread. This prevents scenes from being cloned and game objects from being instantiated without rendering them on the screen, leading to the "blackout" limitation of the state-prediction process discussed in section 6.4.

Chapter 3

RELATED WORK

3.1 Game State Prediction

The vast majority of prediction research in video games has focused on either predicting the player's or the opponent's strategy. The primary focus of video-game AI has been to create challenging and unpredictable AI opponents for a human player, so the ability to identify and react to the strategy of the opponent (from the AI's perspective, either the human player or opposing AI characters) has been a strong motivator [14, 16]. These techniques are often very similar to player strategy prediction techniques (discussed in section 3.2). In many games, however, the opponent is operating with imperfect and limited information of the player's actions or positions, often described in video games as the "fog of war" [44]. This mirrors the player's knowledge of opponent actions, so approaches that tolerate fog of war could be translated to a companion AI that tries to predict and react to the future game state as a human player would.

One example of this that was shown to be relatively successful was in using various machine learning and rule-based techniques to predict player actions in the popular online game, "Starcraft" [44]. These researchers gathered game data from thousands of professional Starcraft matches and used this data to predict when a player in a new game would take an action (creating a specific unit or building in the game), with a fog of war limiting the AI's visibility of the game world to only areas near where the AI had units stationed. It is important to note that they attempted to not only predict the player's next action but also predict when in time it would be taken. The accuracy of their predictions was good, and increased as the game went on and

the similarities between the player's strategy and the pre-recorded game strategies were better identified. This shows that it is possible to accurately predict the player's actions even with a restricted view of the game world, leading to a more human-like (and therefore less frustrating) opponent for the human player.

Predicting opponent position with limited information is an example of a more speed-sensitive task. In first-person shooter (FPS) games, players are required to very quickly predict where their opponents may be while they are out of sight, using information like their last known location and direction of movement, as well as possibly any paths through the game map that are known to be more likely. AI in FPS games often have complete knowledge of player locations regardless of line-of-site (which human players are reliant on); this is often frustrating to the players and seen as the AI "cheating" [26].

An overview of models for predicting an opponent's prediction without relying on omniscience demonstrates that AI can remain relatively accurate in real-time scenarios even with limited information [26]. Researchers used semi-hidden Markov Models as well as particle filters to predict opponent (human player) positions in "Counter-Strike: Source" [12], an online FPS game. While the accuracy was obviously less than an AI given complete knowledge of player position, when their approaches' predictions were compared to human predictions they were found to be more accurate, and when errors were made they were seen as more human-like.

For this project we considered using similar techniques for predicting future game states, but decided in favor of a new technique, detailed in section 6.4.

3.2 Player Strategy Prediction

Researchers have worked on identifying aspects of the player's strategy for years. Often the goal of player modelling is not to predict specific actions and when in

the future the player will do them, but something more simple. The most common example is to use player behavior models taken from a large number of players, compare them to the current player to find a group of others that behaved similarly, and use the other player's performance as a rough predictor of how the current one will likely behave [19]. This can be used to dynamically adjust the difficulty of the game to keep the player engaged without becoming frustrated [27], to predict something about the future game such as how long it will take the player to finish the game [30] or what actions they might take in the future without predicting the specific time at which they'd be taken [25].

The vast majority of methods for identifying the player strategy rely on collecting a large amount of data from a large number of people to find similar strategies to a current player [30, 25, 23]. A simple example of this technique is shown in an experiment where the researchers attempted to make a small set of predictions about a player's performance in "Tomb Raider: Underworld" [30]. In-game data was collected from a large number of players, then as a new person played the game their strategy was analyzed and various supervised learning classifiers (most successfully linear regression and decision trees) were used to predict whether or not they would finish the game, and how long it would take them to do so. Even with only a small set of relatively simple factors to predict, their accuracy was fairly low and the author's concluded that their techniques would not be accurate enough for real-time game adaptation, but could be useful as a source of feedback on the game design.

A more complex example of this strategy was demonstrated by evaluating game data from the popular multiplayer online role-playing game, "World of Warcraft" [25]. Using a large amount of player data the authors were able to identify "cliques" of achievements that tended to occur sequentially together. As a new player performed actions and reached achievements the most likely clique was then identified and a series of their next actions could be predicted. It is important to note that this

technique predicts an ordered series of actions, but not how much time will elapse between any two actions.

This represents a much more complicated prediction than something like determining whether or not the player will finish a game, and would be much more useful for a real-time AI agent. However, due to the amount of data this method requires and the complexity of their predictions the computation time exceeded what would be appropriate for a fast-paced real-time game, and would be better suited to games where predicted actions or achievements span a longer time frame. This would also prevent this approach from being directly translated into a framework, as the framework must be kept generic enough that it could be applied to games regardless of the pace at which they are played.

3.3 Frameworks

Due to the complexity of developing a well-made companion for a video game, it would be useful to create a generic framework to help speed up the integration of AI into a new game [17, 20, 15]. Such a framework would benefit game developers in two ways; the time required to add a companion to the game would be greatly reduced, as would the machine-learning and AI knowledge required to implement the companion, opening up the framework to a larger group of developers [35]. Following are examples of some previously created "Learning by Observation" (LBO) frameworks.

3.3.1 jLOAF

jLOAF (Java Learning by ObservAtion Framework) was developed as an effort to minimize the effort required to create an agent in a video game [20, 22]. It enables agents to learn by observing human players, saving their actions and some information about the game state when the action was taken to use as a guide later on. The

AI then observes the current game state and acts how it thinks the player would have based on the data. This allows the companion to learn when to perform an action without explicit instructions from the developer. However, agents created using jLOAF are purely mimicking rather than complementary, and it does not attempt to keep track of the actual results of an action, nor does it measure the actions' success or usefulness to the player.

3.3.2 MimicA

"Lord of Towers" (LoT) is a top-down tower defense game featuring a companion AI framework called "MimicA" [15]; this framework was used as the basis for the framework discussed in this paper, and LoT was used as one of the game environments for testing our framework. MimicA is a learn-by-observation framework that adds a companion character that mimics the player's decisions. It was chosen in part because the source code was readily available and the game itself was simple but also largely because it does not rely on data from previous gameplay, making it more useful for new games.

Lord of Towers features a set of actions that the player can perform (see table 6.1), and an open map upon which the player can freely move. Waves of enemies enter the game and the player's goal is to prevent them from reaching a central base for as long as possible. Game states are saved whenever the player makes an action and after some time has passed an AI companion enters the game, having attempted to learn the player's strategy from this data. After it enters the game it does its best to mimic the player, choosing actions based on the current game state and how the player behaved in the past. At a set time in the game the player character dies and the companion is left to survive for as long as it can.

MimicA requires a developer to provide a game state consisting of the player's

chosen action and the relative position of all objects currently in play, as well as a list of possible actions. Then, when the companion needs to decide which action to perform it extracts features from the current game state and uses a classifier to predict which action the player might take, based on the state-action pairs generated before the companion's arrival. The developer does not supply the framework with any indicators of what effects an action might have; the companion blindly copies the player's action without knowing what the consequences of the action will be.

3.4 Examples of Companions in Published Games

Many commercial games feature one or more companion characters, with varying degrees of importance to the game. Companions are most common in role-playing games (RPGs) and first-person shooters, but they are present in many others as well. Sports games that simulate real-life team sports, like the NBA [7] and Madden [5] game series, would also benefit from high-quality teammate AI. In most games the companion's behavior is scripted and does not change player-to-player, and could be improved by applying AI techniques discussed in this paper to create a more believable companion. This section examines some popular commercial games and how their companions affect their quality.

More RPGs contain companions than any other genre, often with many unique companions available to the player, like the "Fallout" [4], "Star Wars: Knights of the Old Republic" [10], and the "Mass Effect" [6] series. "Skyrim" [3] is often held up as the standard by which modern RPGs are measured, including the quality of their companion characters. While Skyrim was very well received in general, many reviewers complained that the game's companions left something to be desired; one reviewer complained that the companion AI is "bad and frequently steps in front of you to take friendly fire and just die" [9]. A mimicking companion could help

address these complaints, and a complementary companion could be even more useful. Actions could be identified that assist the player's immediate goals, like healing the player if they're weak or distracting enemies during a fight, resulting in a companion that is more useful and appreciated.

Shooters, such as the Call of Duty [2] and Battlefield [1] franchises, also commonly have a number of companion characters that typically act as the rest of the player's team or squad. Most games focus on using AI for creating interesting opponents, so the majority of shooters have very predictable teammates that act according to a well-known series of scripts. A few games, like "Star Wars: Republic Commando" make the companions the focal point of the entire game, with very good results [8]. In "Commando" the player controls a unit of soldiers, directly controlling the team's strategy as a whole. The AI controlling the rest of the player's squad in Commando is often applauded, because the way the companion's interact with each other to carry out the player's directives is very good. Unfortunately this still relies on the player to dictate the team's strategy very often; companions that decide for themselves how to work together to further a player's strategy could increase the immersiveness and player's enjoyment of the game even further.

Chapter 4

DEFINING "COMPLEMENTARY"

Little research within computer science has been done regarding complementary actions, but a working definition is required to serve as the basis for the companion character's behavior. To create a concrete definition, we studied psychology papers regarding complementary [39] and "pro-social" [41] behavior, as well as "joint-action" [28]. Briefly, complementary behavior have been defined as multiple agents coordinating different actions to further a common goal [39]. Distinctions have also been made between "planned" and "emergent" coordination between agents [28] and for a companion character in a video game, both are required. "Planned" coordination can be encoded by the game developer to keep the companion working towards the overall goal of winning the game, and "emergent" coordination is required to tailor the companion's strategy to each individual player.

It is important to emphasize that complementary behavior is not necessarily imitative [39]. An agent performing an identical action to their teammate may further the group's overall goal, but it is quite possible that the goal would be better served by a different one. In the context of a video game an imitative companion is limited in their behavior; if the player has never performed an action the companion will avoid it as well, neglecting potentially useful actions. However, if the companion does not take into account the player's set of actions at all they may repeatedly do something that the player was intentionally avoiding, negatively affecting their strategy. Therefore, a balance must be struck between the two extremes.

It is also necessary for a complementary agent to attempt to predict the future actions of the player as well as the effects that its actions might have [39]. Without any predictive capability, an agent would be unable to judge how well a given action

meshes with the overall goal. Because a complementary companion is deferential to the player rather than in a traditional multi-agent environment, it also needs a predictive capability to prevent inadvertently doing an action that disrupts the dominant agent's plans. In a video game, the player serves as the dominant agent, and all decisions that are made must prioritize them as much as possible.

These definitions provide a starting point, but they must be slightly adapted for this project because a complementary companion in a video game necessarily has a more subservient role than a traditional teammate. The overall team's goal is not often not as well defined; even when it is it's secondary to the current player's goal, which must be regarded as more important and cannot be set by the developer ahead of time. Therefore, our definition is as follows.

A companion's actions will be considered complementary if:

- They identify the player's strategy in the near-future whenever possible by predicting the player's next actions, removing obstructions to the strategy if possible.
- If there are no obstructions or they cannot be removed, the companion will benefit the player by improving the state of the game in their favor, as determined by the developer (score, etc.).
- They do not jeopardize the player's "near-future" strategy for the sake of other perceived benefits, like an increased score.
- They do not limit themselves to mimicking the player's behavior, but still prefer similar actions to avoid unintentionally jeopardizing the player's strategy.

Chapter 5

SYSTEM DESIGN: COMPLEMENTARY DECISION MAKING

5.1 Overview

This chapter details the overall decision making process used by the companion to identify actions that are complementary to the player's immediate and overall strategy, based on the current game state. The developer's game must initially define a list of possible actions as well as a default action that will be done when the companion has no better options. When a new action is required the companion must call the getDecision() method in the FrameworkCompanionLogic class, passing in a game state vector representing the current state of the game.

The bulk of the decision making process is described in section 5.2; when that process outputs an action it is passed through a final sequence, discussed in section 5.3. A graphical representation of the main and final decision making processes can be seen in figures 5.2 and 5.3, while the combined full flowchart is included in appendix B.1. For quick reference, a simpler version of the full flowchart can be seen in figure 5.1. An example of the companion making a decision is included in section 5.4.

There are multiple steps of the process where one of two branches is taken based on a stochastic decision. In each of these cases the chance that one branch is taken over the other is controlled by the developer, allowing them to fine-tune the parameters to best fit their desired behavior.

Due to how parameterized and customizable the decision-making process is, it's not possible to quantify how often each branch of the tree is taken in a general sense. These statistics depend entirely on how the framework is configured by the game developer, as well as the types of actions present in a game and how the game itself

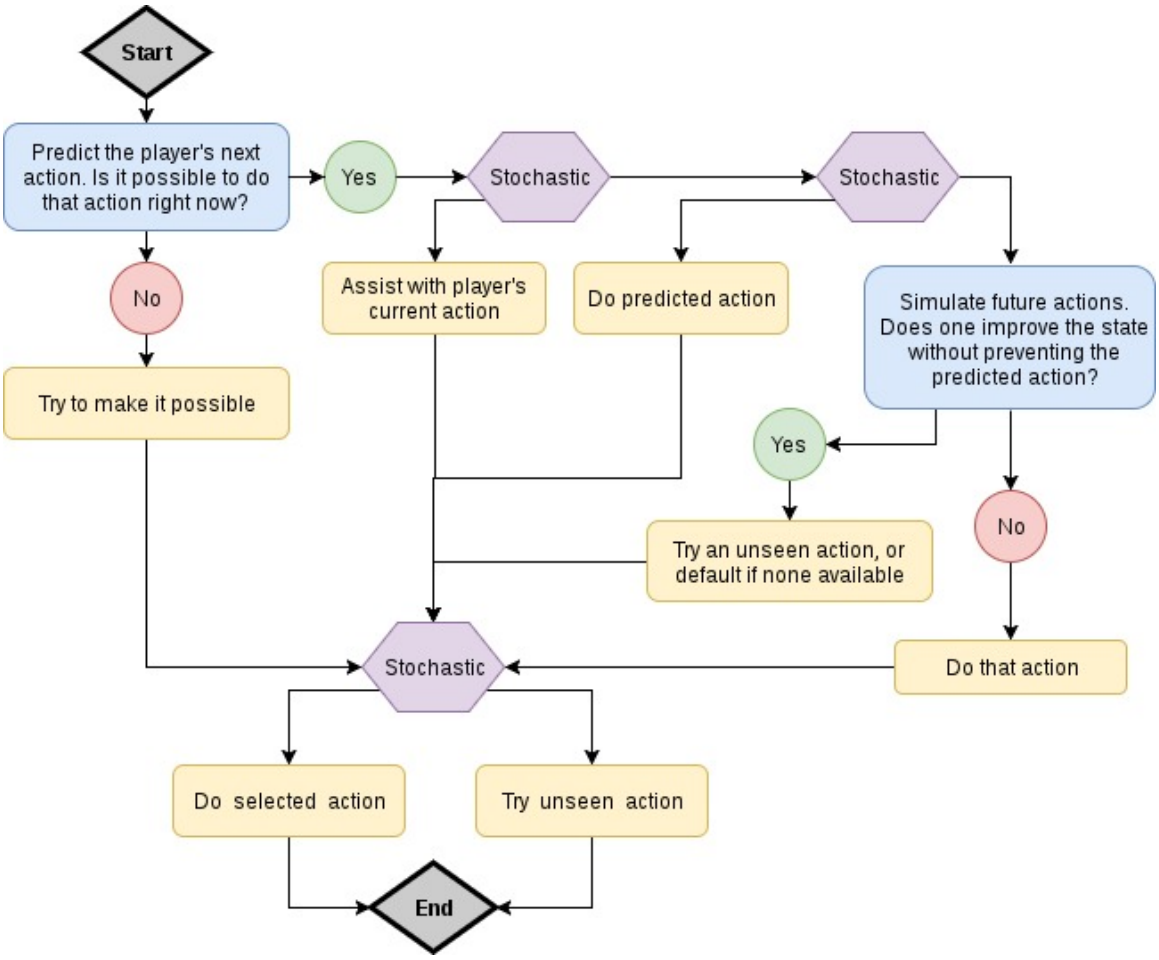

Figure 5.1: A simplified flowchart showing an overview of the complementary decision-making process

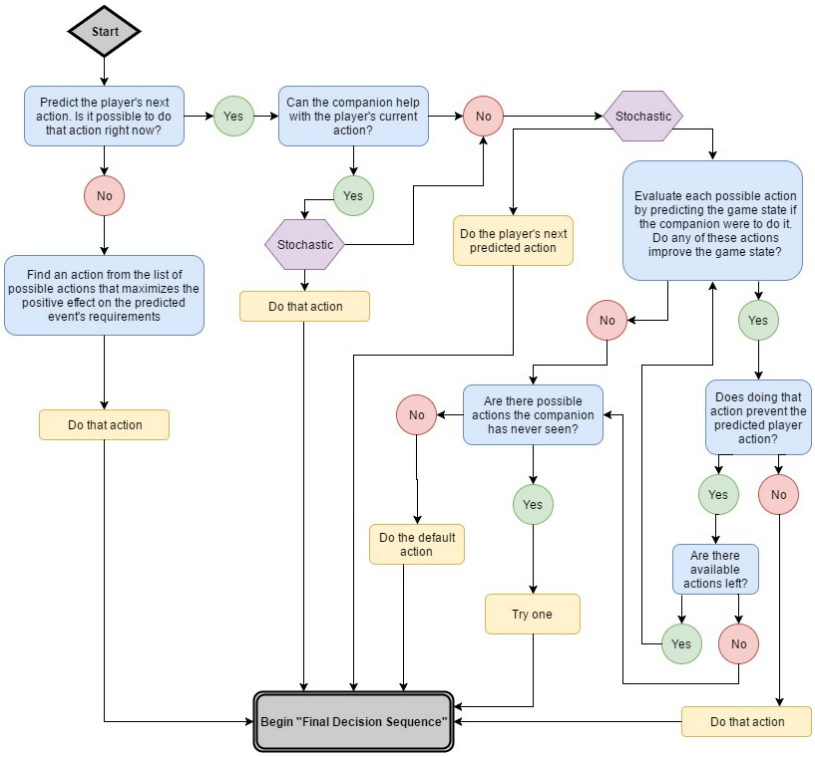

Figure 5.2: The first portion of the decision making process

is balanced. With a slight modification the framework could record this information, which could potentially be used to dynamically reconfigure the inputs to make the branch statistics match a developer-specified goal.

5.2 Main Process

The first step of the decision making process is to predict the action that the player is likely to take after they finish their current action, based on their previous data. This process is detailed in section 6.3. The abstract class FrameworkEvent, which all possible actions must extend, defines a method that determines if an action is possible in a given game state, represented by a game state vector. The companion checks whether or not the predicted actions is possible in the current game state; if it is not, they search for an action that results in a state where the action can be done, using the state prediction process explained in section 6.4.

This choice was made because the companion's fundamental goal is to further the player's strategy. The most basic level of this goal is to prevent situations from arising that prevent the player from doing what they want to do and forcing them to alter their plan. The importance of this goal is why attempting to enable the player's predicted next action is the first step in the companion's decision making process.

If the player's next action is possible in the current game state, the companion next checks if they are able to assist with the player's current action. Each action is required to have a boolean field specifying whether or not multiple entities can perform that action at a time; for example, healing themselves may be a one-character task, while multiple might be able to work together to construct a building more quickly. If the companion can assist with the player's current action there is a chance that they do so without any further deliberation. This was decided because finishing the player's current action more quickly, like preventing their next action from being blocked, objectively helps further the player's immediate goals.

If the companion cannot help the player perform their current action, the companion's decision next randomly chooses one of two paths. One of the two options is for the process to return the player's predicted next action. This was determined to benefit the player's strategy because they likely have a series of actions that they would like to complete and that would need to be done serially if there were no companion present. The companion doing the action for them allows for the series of actions to be done in parallel, reducing the overall time required to complete them and ultimately working toward the player's goal. Again, the likelihood of taking one branch over the other is set by the game developer.

If the other branch is taken the companion enters the state-prediction sequence is explained in section 6.4, similar to what would happen if the player's predicted action had not been possible in the current state. Differing from the first state-prediction

sequence, which evaluated states based on whether or not they allowed the player's next action to be performed, this sequence evaluates them based on the scores of the future states alone. A list of possible actions is returned, ordered by descending state score. The first option is checked to ensure that it doesn't prevent the player's future action from being taken; if it does the next one in the list is checked, if it doesn't it is returned as the companion's next action. This allows the companion to assist the player's overall goal even though they are currently unable to help with their immediate strategy.

If the list of options is exhausted, meaning all of them prevent the player's predicted action, the companion takes a last resort of looking for any actions that it knows are possible but that the player has not performed yet. The state-prediction process only evaluates actions that the companion has seen the player perform, keeping the companion's behavior in line with the player's in case any actions were deliberately avoided. At this point the companion could not do any of those actions without harming the player's strategy so they must choose a random new action. If there are no unseen actions the decision making process returns the default action, specified by the developer. In Lord of Towers, the default action is to wait and do nothing for a set amount of time before trying to find a new action again.

5.3 Final Decision Sequence

The final step of the decision making process is called the "Final Decision Sequence." During this sequence the choice of action is finalized and returned to the developer. It is a simple process; the actions that have previously been performed by either the player or the companion is compared against the list of all available actions. If there are any actions that have never been seen there is a random chance set by the developer that a new action will be tried in a random region (region calculation is

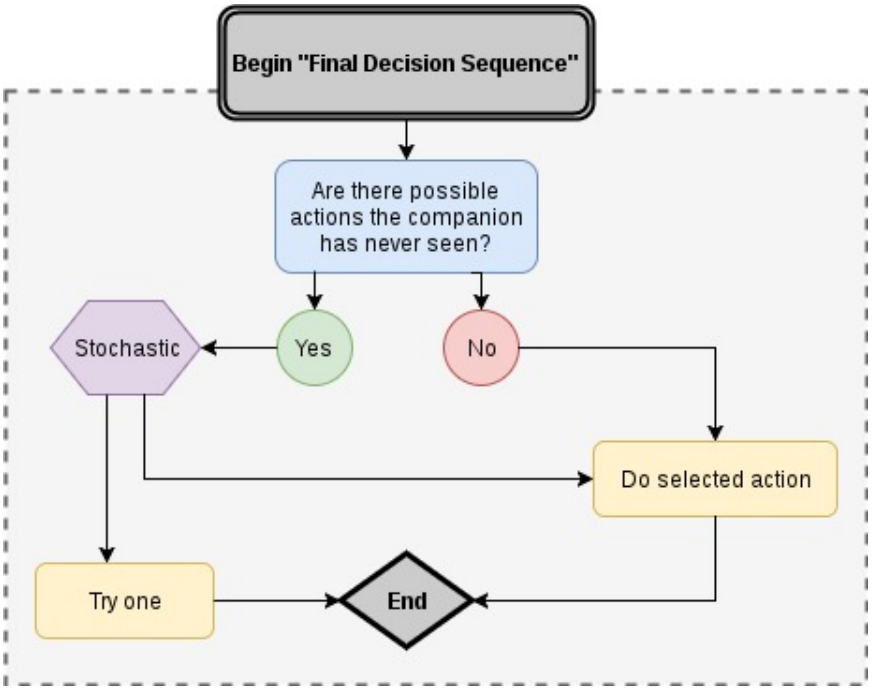

Figure 5.3: The second portion of the decision making process

discussed in section 6.2). A diagram of this process can be seen in figure 5.3.

This step allows the companion to occasionally experiment, ideally exposing the player to strategies that they had not considered. A human teammate would likely not restrict themselves to actions the player has seen; this approach was chosen to make the companion act more realistically. If an unseen action is tried, it is not added to the list of seen actions until the player performs it as well. This prevents unwanted, intentionally-avoided actions from entering the data used during the rest of the decision-making process. For example, if the player was avoiding chopping down trees to keep a natural barrier between them and their enemies, the companion chopping one tree down to get desperately-needed wood would not increase their likelihood of chopping down more.

5.4 Example Decision

Imagine a simple game with three possible actions available to the player and the companion: Fight, Build, and Gather Resources. When the game starts the player begins to perform actions which are recorded, eventually trying every action. Once the companion is introduced, it begins its first decision-making sequence based on the current game state. In this game state, the player is performing Gather to get more resources, which can only be done by one entity at a time.

First, it uses the players recorded data and the current state to predict what the player will do next; it predicts that the player will want to Build. Build requires 100 resources; in the current game state the player has 200, so Build is possible. If it hadn't been possible, the companion would have entered the state prediction process to determine if any action would make it possible, which in this case would be Gather. Because Build is already possible, the companion moves to the next step of the decision-making process. Because the player's current action, Gather, can only be done by one character at a time, the next step is skipped. If multiple people could Gather at once, there would have been a random chance that the companion would have gone to assist the player.

Next, a stochastic chance determines whether or not the player will do the Build action on the player's behalf. In this example, chance determines that the companion should not do this, instead continuing the decision process. Now the companion enters the state-prediction process again, trying each action and finding the one that results in the game state vector with the highest score. Here, the simulation process returns Fight. If doing Fight prevented the player from doing Build the companion would have looked for an unseen action to try, and returned the default action when none were found. Now, the companion enters the final decision sequence with Fight as the chosen action; no unseen actions are available to try, so the companion will Fight.

Chapter 6

SYSTEM DESIGN: IMPLEMENTATION

6.1 Framework

One of the secondary goals of this system is for it to be portable. Originally, MimicA had significant coupling between the code controlling the companion's decision making process and the rest of the game. This made the prospect of porting the framework from Lord of Towers to another game unnecessarily daunting. We modified the existing code to separate the framework from the game, preventing the framework from accessing any game code while providing an easy-to-use interface for the game to interact with the framework. This makes the framework as portable as possible, hopefully allowing other developers to integrate it with their projects. The system is currently closely tied to Unity, but it would be feasible to transition it to a more generic system that could be used in multiple video game engines. The easiest step would be to export it as a C library to be used in other C engines, but it could be ported to other languages as well.

We designed the framework with ease-of-integration in mind, keeping it as easy as possible to either add the functionality to an existing game or build a new one from scratch. A rough outline of the steps to integrate a game with the framework are as follows:

- Define a list of events that the companion and player can perform, extending the FrameworkEvent abstract class.
- Define an extension of FrameworkGameStateVector, which represents the current game world.

- Add an instance of the FrameworkGameData GameObject to the Unity scene.
- Each time the player takes an action, send the current game state vector along with the player's event and location to the FrameworkGameData object.
- Add the "Resettable" interface to any GameObject's that require it (the player, buildings, etc.)
- Create an instance of a FrameworkCompanionLogic object; this will be the companion's "brain."
- When your companion needs to make a decision, call the getDecision() method in FrameworkCompanionLogic. This will return an object that contains a list of events to perform, along with a FrameworkRegion object that determines where in the game the action should be done.
- Determine where in the given region to perform the action, perform it, and repeat.

A more detailed description of the steps required to integrate the framework into a game is provided in appendix D.

6.1.1 Lord of Towers & Lord of Caves

Lord of Towers was originally developed to test MimicA. It's a simple tower defense game, where the player can build and upgrade defensive buildings (walls, towers that shoot at enemies, etc.) to defend a castle from incoming waves of enemies. Periodically, companion characters enter the game that attempt to mimic player behavior based on the current game state. After a fixed amount of time the player's character dies, and the companions try to continue the player's strategy to survive as long as possible.

Some changes to the game were required because this framework's purpose differs

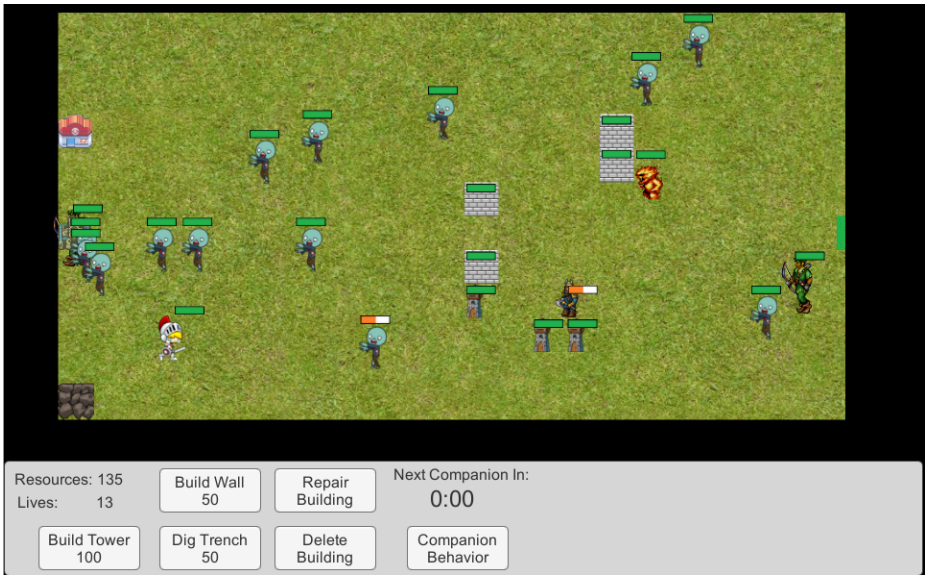

Figure 6.1: An example of gameplay in "Lord of Towers"

from MimicA's. Because the companion's actions in relation to the player are the primary focus we no longer force the death of the player character, instead ending the game when the player dies or the castle falls. We also limited the number of companions to one at a time due to technical infeasibility. This is discussed further in the "Future Work" section of the conclusion. A new action, "Mining," was also added, offering players a different method of generating resources other than the deaths of enemies. We also made general improvements to the game's artwork. Table 6.1 shows a list of the possible actions in Lord of Towers. Lord of Towers gameplay is shown in figure 6.1.

Table 6.1: Events in Lord of Towers

Event List in Lord of Towers		
AOE Upgrade	Heal Themselves	Repair Wall
Build Tower	Mine	Slow Upgrade
Build Trench	Move	Upgrade Tower Ability
Build Wall	Range Upgrade	Upgrade Tower Fire Rate
Damage Upgrade	Repair Tower	Wait (Default)
Delete Tower	Repair Trench	

To demonstrate the system's portability we developed a second game alongside

Figure 6.2: An example of gameplay in "Lord of Caves"

Lord of Towers, called "Lord of Caves." Lord of Caves was an integration of Lord of Towers and a separate project on constrained procedural map generation for 2D games. It has similar gameplay to Lord of Towers and shares much of its artwork, but the code-base is separate and demonstrates the portability of the new framework. The genre is different as well; rather than defending their base for as long as possible the player must search for and destroy an enemy base instead. In Lord of Caves, the player is placed randomly on a generated map, and a fog of war prevents them from seeing areas that they have not explored. An enemy castle that spawns zombies is also placed on the map, and the player's objective is to fight their way to the castle and destroy it. They can build towers like in Lord of Towers, and they can find a companion hidden elsewhere in the level that joins them. An example of a map in Lord of Caves is shown in figure 6.2.

6.2 Dynamic Region System

Lord of Towers was originally implemented with a fixed-size map split into six equally-sized sectors. The player's actions were saved along with the sector that in which they occurred, and this data was used in MimicA [15] to train the companion. When the companion chose an action to perform it returned an action-sector pair and Lord of Towers determined where in that sector to perform the action. By returning a sector rather than a global coordinate, the framework allows the game designer to fine-tune the result in a way that may make more sense within the context of their game.

This posed a problem with respect to the companion's choices. Static and predetermined sectors severely limit the granularity of the companion's choices; for Lord of Towers in particular, players have a strong tendency to perform the majority of their actions in the sectors closest to the tower. This effectively lowered the number of different sectors present in the data from six down to two or three. This limited the usefulness of the framework's location decision and offloaded a large amount of the decision-making process to the game. From a developer standpoint, defining useful regions for a potentially large number of maps is tiresome; a dynamic region system would alleviate this burden as well.

An alternative, dynamic region system was adopted to address this issue and give more power to the framework while still leaving the details to the developer. In this system the map is initially set to one large region rather than six sectors. As the player takes actions they are stored in the framework along with their global position, rather than which region or sector they occurred in. Along with recording the player's action, new regions are created with each action the player takes. The region containing a new action is split in half along it's longest axis, and the global list of regions is updated accordingly. Figure 6.3 shows an example of how a map's regions could be created; the map on the left shows the region layout after the first

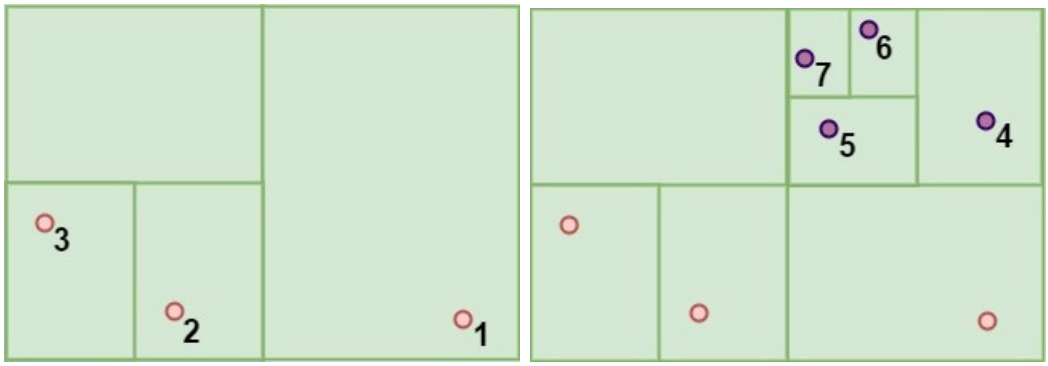

Figure 6.3: An example of the dynamic region system. The first image shows the regions after the first three player actions, where the numbers indicate the order. The second shows how the regions are updated after the player performs four more actions.

three actions, and the map on the left shows it after four more. When the companion makes a decision it chooses a global coordinate based on those recorded from the player, which is translated to a region based on the most up-to-date distribution before the decision is returned to the game.

This results in a constantly updating set of uneven, rectangular regions that is customized based on the behavior of the current player rather than being fixed at the beginning of the game. As the game progresses and more regions are created their sizes become smaller and smaller; a limit to the minimum area of a region can be set if the developer wishes, but in the case of Lord of Towers the tiled map imposes a minimum size without the need to explicitly specify one. The benefit of this system is that the regions returned from the framework have much more usefulness than the previous large sectors, but still allow the game designer the leeway to decide where exactly within a region an action should take place. It also adds very little overhead; identifying which region contains a global coordinate is trivial, and when a new action is taken only the relevant region is changed without recalculating the rest of the list of regions. It also lends itself well to choosing a random region, something that happens when previously-unseen actions are tried; an areas that the player has focused their

actions more densely in will be broken up into more regions, so a randomly chosen location will follow the same distribution of the player's actions and be more likely to be in a similar place. This still allows for the companion to experiment because the larger, less populous regions can still be chosen.

6.3 Player Action Prediction

Prediction over the player's likely next action is handled using a decision tree classifier trained on the player's previous data. The data requirement prevents the game from introducing the companion until enough data has been recorded, so the companion cannot be present when the game starts. There is not a fixed amount of data that is considered "enough;" this is decided by the developer, and more data will result in more accurate predictions. The data is stored as an ordered list of pairs of game state vectors and events with a global coordinate attached that indicates where on the map the player took the action.

When the classifier is trained (either the first time it is needed or when the developer requests a retrain), a secondary data set is built from the player's data and used to fit the classifier. By default the entire game state vector is used but the game developer has the option to choose from a number of ways to manipulate this data, specified by an Enum passed to the classifier. These are as follows, and could be expanded in the future: use the entire vector (the default), use only the previous action, use the previous three action, or use the previous five action. This allows the game designer some further leeway to choose how the data is used, in case something about their full vector makes it a bad data source in its raw form. A method is provided to try multiple trials of each on a set of player data, returning the method that had the highest average accuracy. For Lord of Towers, we used the full data vector.

Once the classifier is trained the current game state vector can be used to predict

an action-location pair. Rather than return the location to the developer as a single point in global coordinates, the region containing the point is returned based on the current global set of regions (see section 6.2). This is then used later in the companion's decision-making process.

6.4 Game-State Prediction

Multiple parts of the companion's decision making process require the companion to predict how each possible action will affect future game states, choosing an action based on which results in the best foreseen outcome. To facilitate the comparison of the predicted game states the FrameworkGameStateVector class specifies a method that returns a numerical score; once a mapping of possible events to predicted vectors is created, the action with the highest-scoring vector is chosen.

To build the mapping of action to vector, we chose to have the companion actually perform each action and then wait for a given amount of time to allow the effects of the action to be felt. Initially we planned on using a traditional machine learning technique to predict future game states; particularly, linear regression. This would have limited the game states to only numeric data, which we decided would limit the developer too much. Furthermore, the accuracy of any machine learning method seemed too low, particularly with the likely small quantity of data available. To avoid these issues we chose to have the companion perform each action and calculate the actual game state's score, rather than predict it based on previous data. We implemented a state-saving and loading process, so the state can be saved before the prediction and reset before each new action. To integrate the state-saving process into the game the developer must simply add the Resettable interface to any relevant object, which will handle the saving and resetting of the object as necessary. How long the companion waits after trying each action is set by the developer. Games

where the effects of actions are not felt for some time should have longer wait times to increase the usefulness of the simulation. A balance between simulation-quality and efficiency needs to be struck, because increasing the time the companion waits also increased how long the entire set of simulations takes to complete.

These predictions would ideally be performed in a separate thread, as they can take a few seconds to complete, particularly with the UI updating. Unfortunately, Unity does not allow for the GameObject methods to be called from a separate thread, limiting us to trying the actions in the visible main thread. This limitation is discussed further in section 10. However, Unity does allow for the speed of the game to be increased up to one hundred times the base rate. Using this, the state prediction process goes as follows:

1. The companion decision-making process enters the state-prediction phase.
2. The game state is saved, and the time scale is increased to it's maximum speed
3. Get the list of previously-seen events from the FrameworkGameData object, and the list of regions from the state-prediction location determination system discussed in section 6.4.1.
4. Return the first event-region pair to the game, along with a "wait" action. Only return action-region pairs where the action has been determined to be possible in the region.
5. Wait for the companion to request another action.
6. Record the new game state (paired with the previous action), reset the game state and return the next action-region pair.
7. Repeat until all events have been tried in all possible regions.
8. Identify the highest scoring action, reset the game and the time scale, and return the best action.

6.4.1 Location Determination with "Safe Regions"

The maximum number of locations the companion tries each event at during the prediction process (N) is determined by the game developer. The map is then split into that number of smaller regions. These regions will be referred to as "safe regions" to differentiate them from the global regions discussed in section 6.2, because they limit the companion's actions to a tighter area around where the player has performed actions. The global region system results in a much larger number of regions than required for the state prediction process; trying each action in each global regions would extend the length of the prediction time to too great an extent. This system is similar, but simpler. First, a single safe region is created that borders all of the player's previous action locations. This region starts a process where the list of current safe regions, starting with just the one, is progressively grown until the required number of regions is met. To grow the number of safe regions, the one in the list that contains the most player actions is split in half, increasing the total number by one. An example of this process is shown in figure 6.4.

The abstract class that all actions in a game must extend specifies a method that returns whether or not an action is possible at a given time and place, specified by a game state vector and a region. Each action is taken in each of the safe regions where it is possible, so the number of action-region pairs per action is capped at N but could be zero if the action is not currently possible.

This is preferable to the global region system for state prediction for multiple reasons. First, it results in a precise number of regions that still approximate the physical distribution of the player's actions, ensuring that the companion's actions remain synced with the player's overall strategy. Furthermore, because the initial region is trimmed to border the player's actions the companion will not choose a location that is drastically far from where the player has acted previously. The global

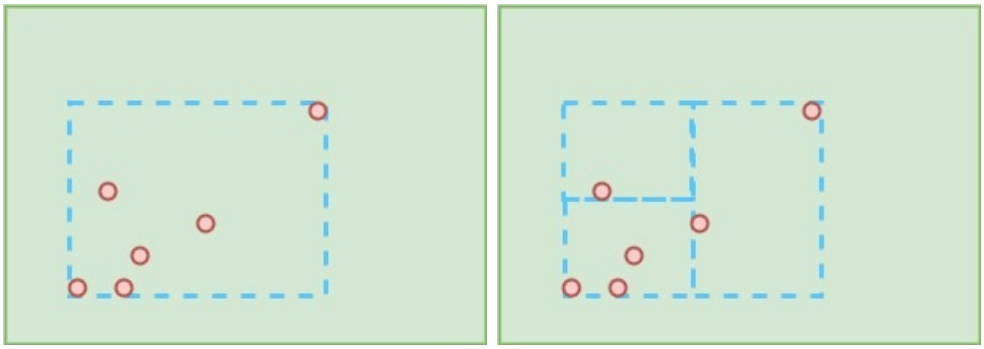

Figure 6.4: An example of the state-prediction region system with N = 3. The first image shows the initial region, trimmed to border the player's actions. The second shows the finished regions after the splitting process is complete.

region system allows for more experimentation, whereas during the state-prediction process we determined that it would be more beneficial to try events in a tighter group that more closely matches the player's behavior.

Chapter 7

EXPERIMENTAL DESIGN

7.1 Methodology

We conducted a user study of one twenty-five person class of undergraduate college students in a intro-level video game design class. The class was divided into two groups, half tested this project first and half testing another project, also based on Lord of Towers. The group testing this project first played Lord of Towers with our companion for twenty minutes total, repeating the game when they lost as many times as possible within the given time. After the time limit, they filled out a survey asking for feedback, then the halves switched and the process repeated.

Participants were not aware that the focus of the project was the companion AI, and the survey given afterwards was designed to mask that fact for as long as possible. Questions asking for general feedback (strengths, weaknesses of the game, etc.) were asked before more specific ones relating to the behavior of the companion. Eventually, they were given a description of the project, the framework, and the companion's decision making process and were asked questions regarding how useful the framework would be to them as game developers. The questions on the survey were focused on the following topics:

- How much they enjoyed the game, and their familiarity with Tower Defense games.
- The perceived strengths and weaknesses of the game.
- General feedback on the AI in the game
- The behavior of the companion

- The potential usefulness of the framework to developers (they were given a description of the framework).
- The effect the blacking-out of the screen during the prediction process on the gameplay and their other opinions.
- An open-ended section for general comments and suggestions was also included.

The full text of the questions is included in appendix A.

7.2 Qualitative Question Coding

This section specifies how the answers to the free-response will be grouped when the results of the survey are evaluated in section 8.

Answers to the questions about the general strengths and weaknesses will be grouped into the one of three categories. Responses that mention the companion will be in one category, ones that mention the gameplay mechanics in another, and all other comments in an "other" category. The questions asking about noteworthy aspects of the AI and what stood out about the game as a whole will be categorized similarly, but broken into five groups because the question allows for both positive and negative comments. There will be a category for positive comments about the companion and one for negative, another pair for positive/negative comments about the gameplay, and an "other" category for the rest of the responses.

In the section asking questions specifically about the companion, the question asking the participants to describe the companion's behavior will be grouped into positive, negative, and neutral comments. The question asking participants to guess how the companion was programmed will be broken into five groups. Any response that indicates that the companion learned from the player will be in one category, and any that mention scripted behavior will be in another. Answers that mention

the companion acting randomly will be grouped separately, and the rest will be in an "other" category. The question asking how the Lord of Towers companion compares to other tower defense games will be grouped based on the differences in the returned results.

The free-response question in the section about the effect of the blackouts during the state-prediction process that asks how the participants earlier answers might have changed will be grouped into two groups. Answers that indicate that earlier answers would have been more positive will be in one category, the rest in another. The final question, asking for general feedback and suggestions, will be categorized after the results have been examined.

Chapter 8

RESULTS & EVALUATION

Overall, the results of the user study were very positive. This section breaks down and evaluates the results of each set of questions asked to the participants. For a brief overview, a summary of sentiment for the free response questions and the multiple-choice question is provided in figure 8.1. This shows the number of unique participants that made comments of particular sentiments in any of the open-ended questions in the survey. Any comment that mentioned that the companion was useful, human-like, or seemed to learn from the player's strategy was recorded as positive and complaints about the companion were recorded as negative.

A number of people also commented that the companion seemed to act randomly, and these were counted separately from the other categories. We attribute the number of comments about the companion appearing to act randomly to the relatively small set of actions that the companion can perform which may have made it hard to determine its motives. A number of players also mentioned that they were unable to play too close attention to the companion during the gameplay because they were absorbed with their character.

For reference, the full text of all questions is provided in appendix A. No significant difference was seen for any questions in the survey between the two groups of students (those that played this version of Lord of Towers first versus those that played it second). Figure 8.2 shows an summary of the total sentiment counts of companion-focused responses across all questions, broken down by group. Group 1 played this version of Lord of Towers first, and Group 2 played it second. The ratio of positive to negative responses (2.8:1 and 2.4:1 respectively) are very similar, implying that there were no large differences between the two groups overall. It should also be noted

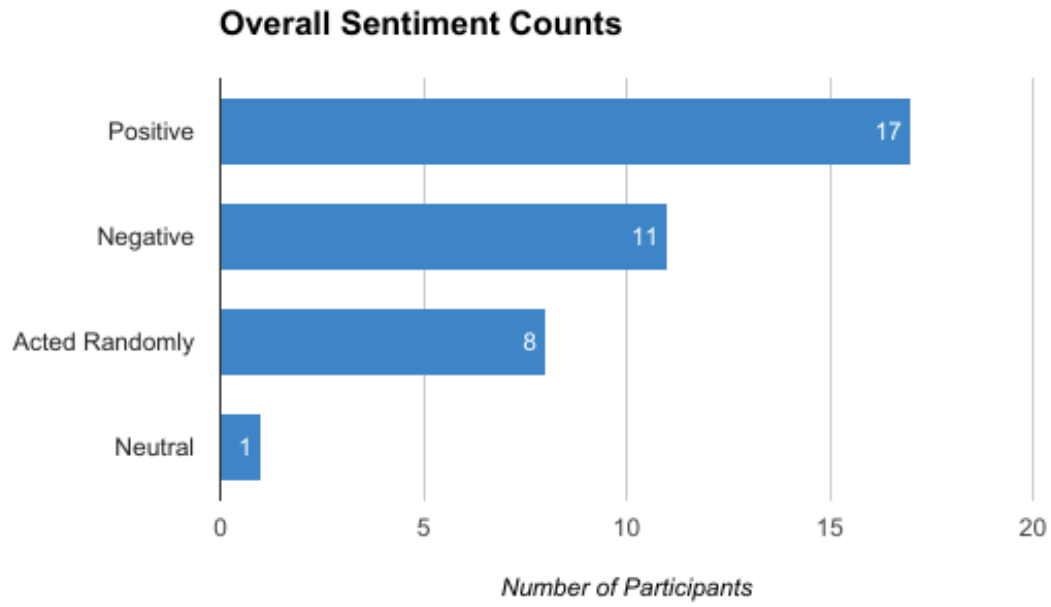

Figure 8.1: A summary of answer sentiments to free-response questions and the multiple-choice question regarding the behavior of the companion AI

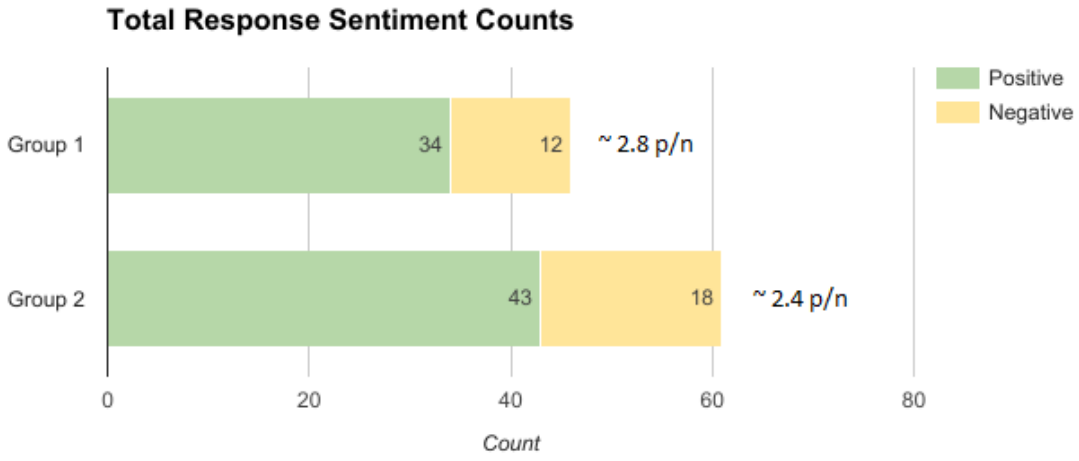

Figure 8.2: A summary of companion-focused response sentiments across all questions, split by group. The numbers at the ends of the bars represent the ratio of positive to negative comments in that group.

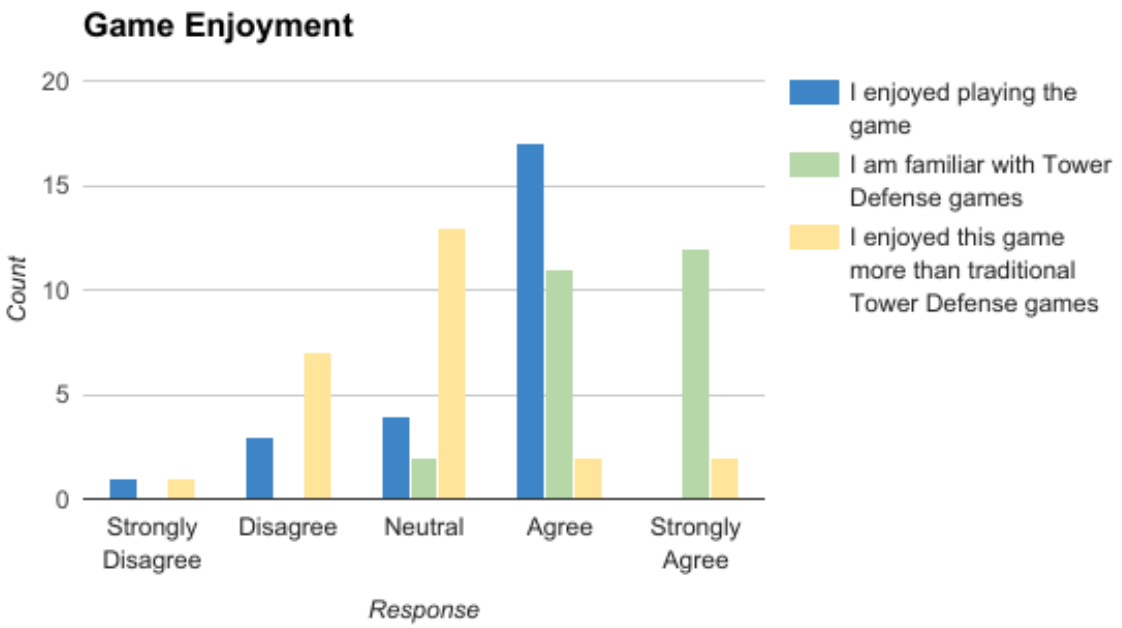

Figure 8.3: A summary of responses to questions regarding participant enjoyment of the game

that a small number of participants had technical difficulties that caused the game to frequently crash; these problems had not been encountered during development and likely had a very negative effect on the participants' responses.

8.1 General Questions

The first set of questions focused on their overall enjoyment of the game. The majority of the participants enjoyed playing, although not (on average) more than most tower defense games. However, as it was not our intent to build a production-worthy game but to showcase the companion behavior, these results are satisfactory for our purposes. A summary of the participants responses to the questions regarding their enjoyment of the game as a whole is shown in figure 8.3.

Regarding the free-response questions about the games strengths and weaknesses, a number of participants singled out the companion. The results of these questions,

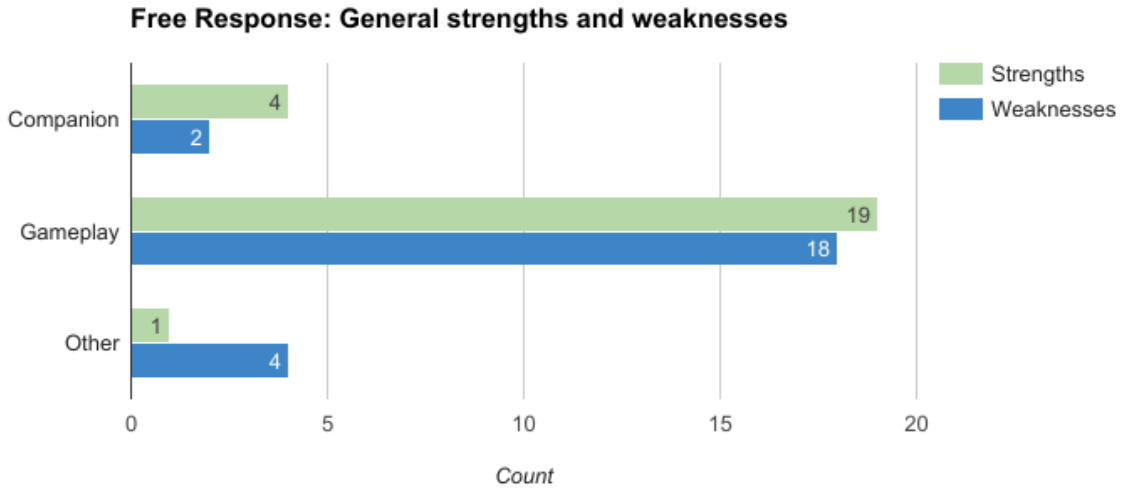

Figure 8.4: A summary of the results of the question about the game's general strengths and weaknesses

Table 8.1: Example responses for each classification for the strengths (S) & weaknesses (W) free-response questions

CLASS	EXAMPLE RESPONSE
(S) Companion	"The companion AI was pretty good and helpful in most cases"
(S) Gameplay	"Upgrading was cool and varying enemies required different setups"
(S) Other	"Having to make decision under pressure"
(W) Companion	"Screen would go black occasionally. Lives would then rapidly fluctuate up and down. Had no idea what was going on. Not sure what companions did as they did not seem to do anything."
(W) Gameplay	"Takes far too long to construct towers/emplacements"
(W) Other	"Needs more effects"

split into the categories outlined in section 7.2, are shown in figure 8.4. Example responses for each type of response classification are shown in table 8.1. Most of these responses were unrelated to companion AI (focused mostly on the gameplay), but four of the participants singled out the companion as a strength, writing comments such as "the companion AI was pretty good and helpful in most cases" and "the companion would actively help you." This is encouraging, showing that the companion behavior

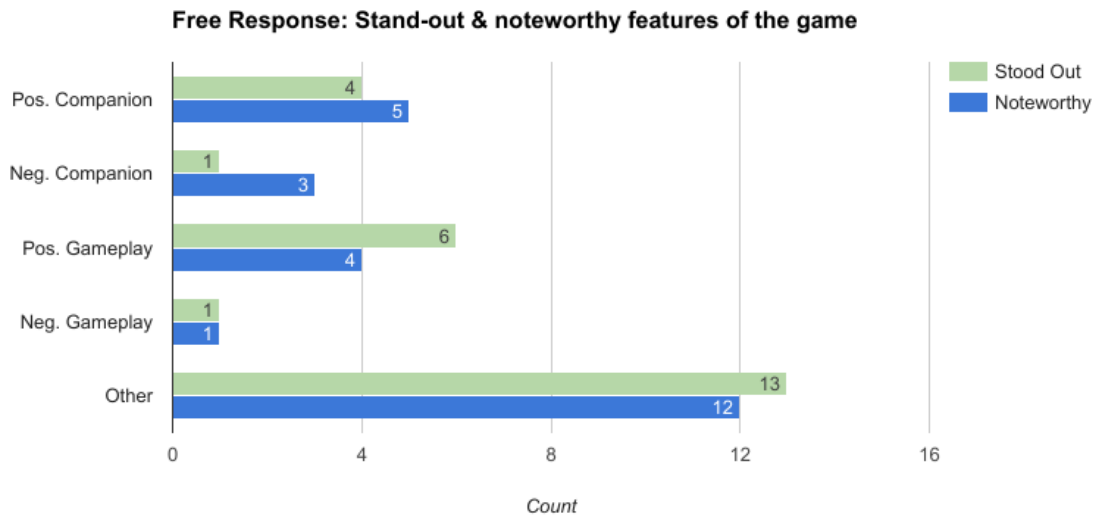

Figure 8.5: The results of the questions about stand-out features of the game and noteworthy features of the AI

was of a high-enough quality to be noteworthy before singling out the companion in more specific questions.

Table 8.2: Example responses for each classification for the stand-out (SO) & noteworthy (N) features free-response questions

CLASS	EXAMPLE RESPONSE
(SO) Pos. Comp.	"Companions! They were smarties."
(N) Pos. Comp.	"The AI actively tried helping me and would mine"
(SO) Neg. Comp.	"Not sure what companions did as they did not seem to do anything"
(N) Neg. Comp.	"Could have been better"
(SO) Pos. Game.	"The generous amount of starting resources"
(N) Pos. Game.	"The path finding for the enemies is cool"
(SO) Neg. Game.	"Skipping ahead. Didn't like it."
(N) Neg. Game.	"Stupid black outs"
(SO) Other	"There was not a consistent art style to the game"
(N) Other	"Didn't have a chance to see the AI much due to time constraint"

The other two open-ended questions in this section asked participants to list anything that stood out about the game as a whole or was noteworthy about the behavior

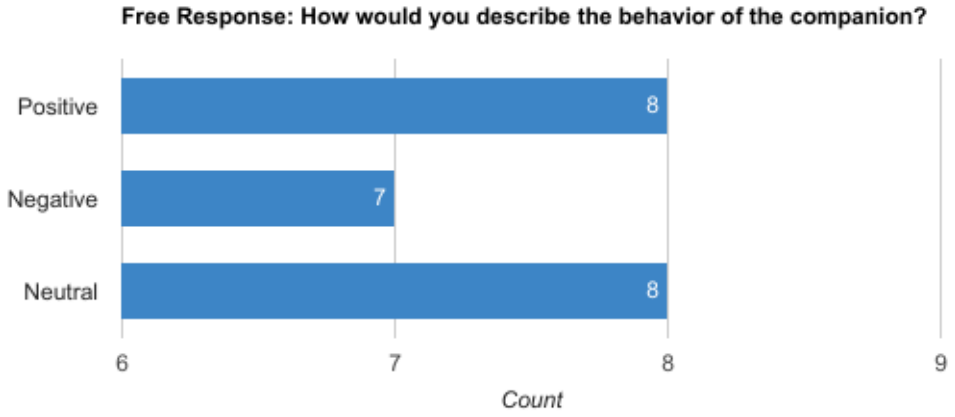

Figure 8.6: The results of the question asking participants to describe the companion's behavior

of any AI in the game. A summary of the answers to these questions is shown in figure 8.5, categorized as specified in section 7.2 with example responses shown in table 8.2. These results were also generally positive, and there were twice as many comments mentioning the companion favorably than mentioned it negatively.

8.2 Companion Behavior

Table 8.3: Example responses for each classification of the companion-behavior free-response question

CLASS	EXAMPLE RESPONSE
Positive	"What I would expect from a friend playing the game (like a slower version of a human player)"
Negative	"Stupid. He messes everything up!"
Neutral	"I'm not sure what to say. I did not pay much attention to them."

The next section asked questions specifically about the behavior of the companion character. The first question asked participants to describe its behavior, and the results of this are shown in figure 8.6. Example responses are given in figure 8.3. These were grouped according to sentiment, and the results are very evenly spread

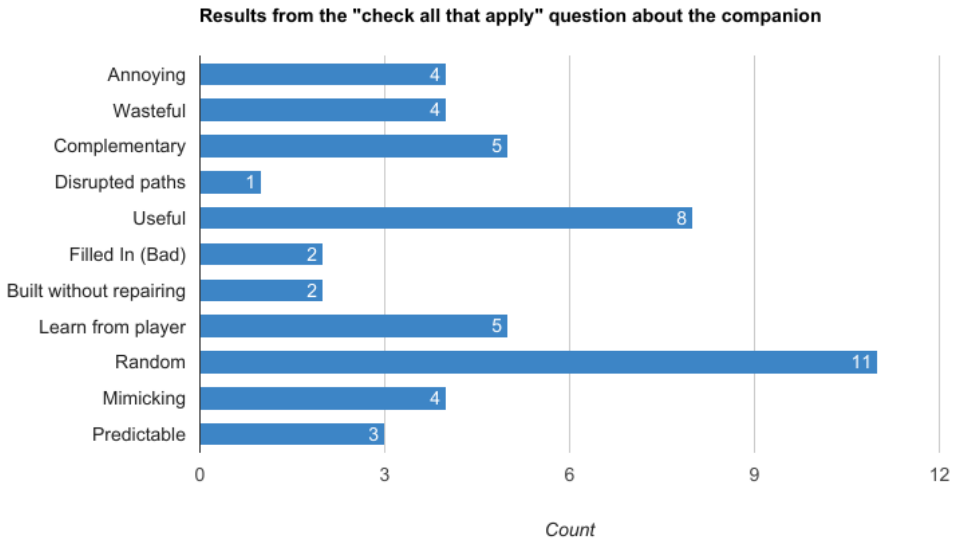

Figure 8.7: The results of the "check all that apply" question about companion behavior

between the three categories. The results for this question lacked the same positive skew that later questions in this section have. This might be able to be explained by the participants not understanding what exactly the question was asking and being confused; if the experiment were to be conducted again this question would be either reworded or asked at a later point of the survey.

Two participants even noted that the companion appeared to act how they would expect a human teammate to behave, saying "[it behaved how] I would expect from a friend playing the game (like a slower version of a human player)" and "it felt almost like half of a player. His actions were as beneficial as my own (to some extent) and did not feel like a pet or random NPC that you'd usually ignore." This is very promising, as one of the goals for our framework was to create believable companion characters.

The next question presented the participants with a number of descriptions of the companion and asked them to check all of the options that were in line with their experience. A summary of these results is shown in figure 8.7. Many of the options

were chosen by a similar number of play-testers, with the obvious stand-outs being the "the companion was useful" and "the companion was acting randomly" options. Again, we attribute the high number of responses indicating random behavior to the relatively small number of actions available to the companion.

The rest of the results were relatively positive, with a high number of participants saying that that companion was useful and more users saying that the companion's actions complemented theirs and that it learned from their actions than the other more negative options. A small group of users complained that the companion was annoying and wasted their resources, but a number of these users also expressed a desire to be able to tell the companion to only gather resources, leaving all other actions to the player. This may have colored their opinion of the companion, as it is possible that they dislike the idea of a fully autonomous companion character.

The disparity in player opinions of the companion's behavior is interesting. While most participants thought that the companion was helpful, some viewed their actions as wasting resources or not doing the actions in a helpful way. We assume that the strong differences in opinion may be a result of how the companion's behavior differs player to player. It is possible that certain play-styles, at least in Lord of Towers, were better suited to the framework than others and led to more useful actions. This could be remedied by incorporating player feedback in-game or having a base level of scripted behavior that constrains the actions returned by the framework. A number of players also mentioned that while they found the companion's actions helpful, they would have liked to be able to constrain the set of possible actions in-game (limiting the companion to repairing buildings, for example). This further suggests that a method for the player to interact with the player could be useful; as the framework already supports being given a list of possible actions separate from what the player can do, this would be trivial to implement on the game-specific side.

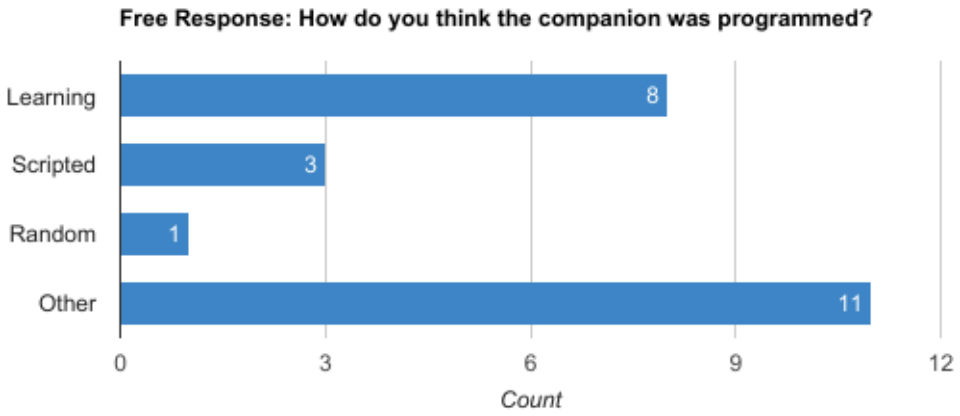

Figure 8.8: The results of the question about guessing how the companion was programmed

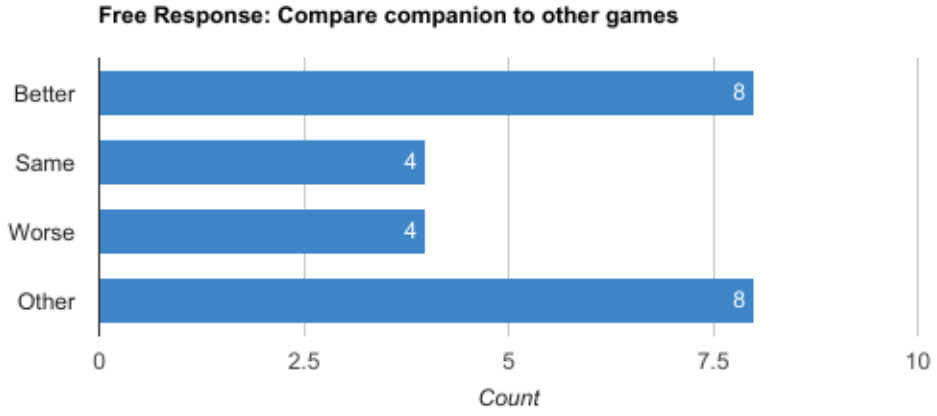

Figure 8.9: The results of the question about comparing this companion to others

Table 8.4: Example responses for each classification of the free-response question about how the companion was programmed

CLASS	EXAMPLE RESPONSE
Learning	"Check patterns that you build in, then try to assist in said patterns"
Scripted	"I think the AI was programmed to target enemies"
Random	"It randomly moves around sometimes"
Other	"Can't really say. didn't get to see much of them on each play-through"

Table 8.5: Example responses for each classification of free-response question about comparing the companion to other games

CLASS	EXAMPLE RESPONSE
Better	"This one learns as opposed to already having a set behavior"
Same	"Uh, pretty equivalent. Killed lot of stuff, able to survive fair amount of attacks."
Worse	"Less useful"
Other	"I haven't played a lot of games with companions"

The final questions in this section asked the users how they guessed the companion's AI was programmed and to compare these companions to the companions in other games. The results of these questions are shown in figures 8.8 and 8.9, and example responses are shown in tables 8.4 and 8.5. The "other" answers mostly indicated the participant being unsure or leaving the answer blank. Both questions had mostly positive answers, showing that the framework has promise and can potentially improve upon the current norm for companion characters.

A number of participants mentioned that they appreciated the companion mining resources for them while they built defenses. This was a side-effect of the prediction processes that we noticed while the framework was under development as well. Because the player tends to perform resource-spending actions and the companion predicts their next action accordingly, the companion often chooses to mine to ensure that the player's next action is possible. While this action is new to this version

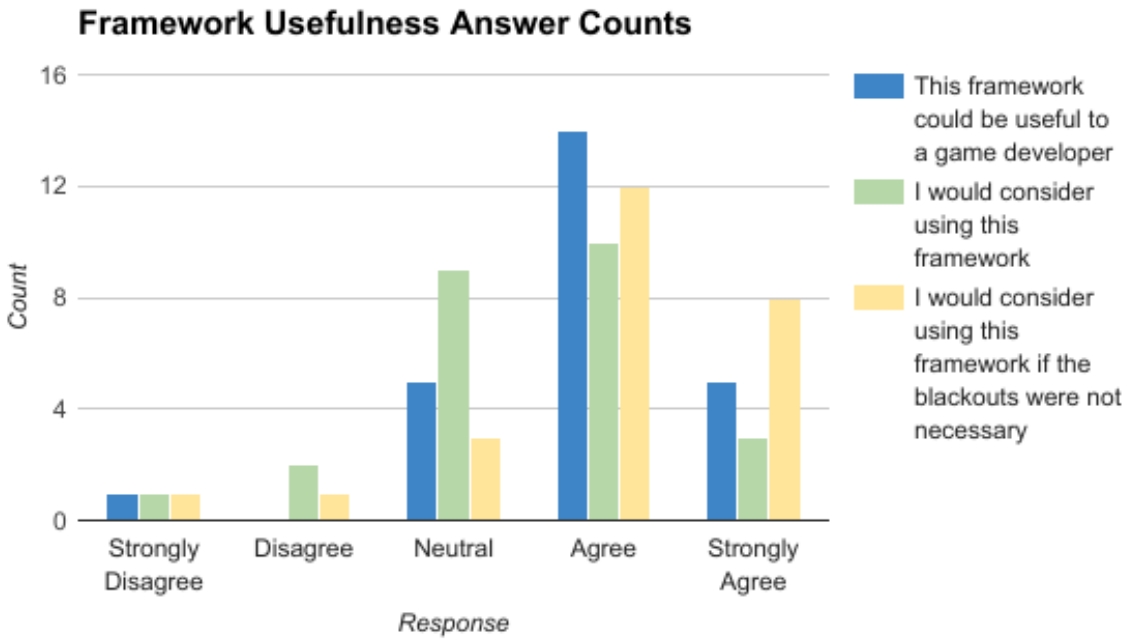

Figure 8.10: A summary of responses to questions regarding the usefulness of the framework

of Lord of Towers a mimicking companion would not have prioritized this to the same extent, instead mining about as often as the player does and leading to more complaints about wasted resources.

Regarding the new dynamic region system for location decisions, the feedback was positive as well. A number of participants noted that the companion seemed to build in helpful locations, leaving comments such as "the AI built towers where I would have," and that the companion would "check patterns that you build in, then try to assist in said patterns."

8.3 Framework Usefulness to Developers

The participants were then given a description of the framework and a summary of the decision making process and asked questions about its perceived usefulness to them as game developers. The results of these questions are summarized in figure 8.10.

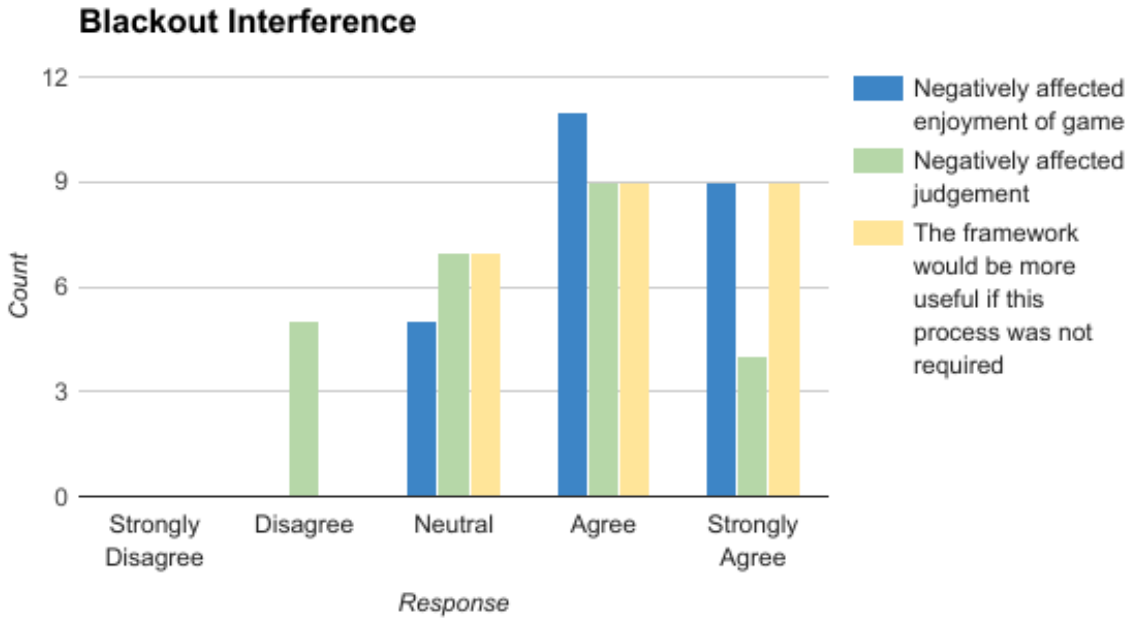

Figure 8.11: A summary of responses to questions regarding the effect of the state-prediction blackouts on participant impressions of the game

Most participants reacted favorably, indicating that the framework could indeed be useful in a general sense. The majority of participants also indicated that the largest weakness of the system was (as expected) the blackouts during the state-prediction process, and that without that the framework would be more appealing.

8.4 The "Blackout" Limitation

Table 8.6: Example responses for each classification of free-response question about comparing the companion to other games

CLASS	EXAMPLE RESPONSE
Improve	"Yes, I feel like the blackouts and skipping of time threw me off a lot"
Not Improve	"I do not think they would have changed"

Because we expected the blacking out of the screen during the state-prediction process to adversely affect the user's opinions on the game, the final questions of the

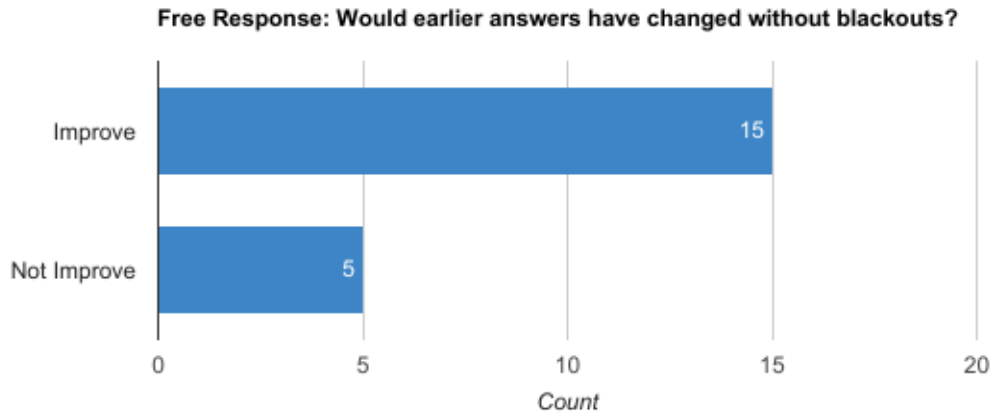

Figure 8.12: A summary of responses to the question asking if removing the blackouts would change their previous answers

survey regarded this issue. The answers to these are summarized in figure 8.11, and most of the participants agreed that the blackouts caused problems with gameplay and the usefulness of the framework as a whole. An open-ended questions also asked the users if and how their earlier answers might have been different had the blackouts not been present. The results of this question is summarized in figure 8.12, and example responses are shown in table 8.6. It should be noted that the participants are likely unable to accurately predict whether or not their answers would have been different, but the results of the question were so strongly in favor of a positive change that it is likely that earlier answers would have improved at least slightly.

8.5 Final Questions

The final questions of the survey asked for general suggestions for the project as well as any other comments. Most of these focused on improving the gameplay (game balancing, improving the artwork, etc.) or general compliments of the project. Only one participant recommended more work be done on the companion AI at all; the rest of the participants mentioned that the blackouts were confusing and that they viewed them as a glitch in the game rather than an intentional feature. This further

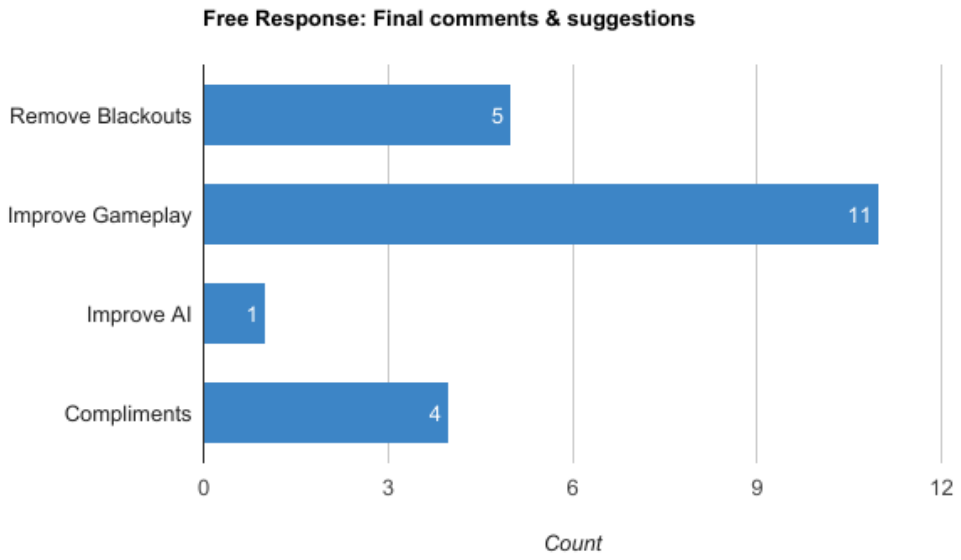

Figure 8.13: A summary of responses to the final survey questions

Table 8.7: Example responses for each classification of free-response question about comparing the companion to other games

CLASS	EXAMPLE RESPONSE
Remove Blackouts	"Get rid of blackouts. Upgrade art assets, would go a long way."
Improve Gameplay	"Maybe add a feature that shows the range of the towers when you go to place them"
Improve AI	"I like the game idea, but the AI is a place that could use some more work. It was really fun though!"
Compliments	"Great game!"

solidifies our conclusion that the results of the other questions would have improved in at least some cases had the blackouts not been present. A summary of the answers to the final questions is given in figure 8.13, and example responses are provided in table 8.7.

Chapter 9

CONCLUSIONS

9.1 Summary

We propose a game-development framework for adding novel complementary companions to video games. A complementary action was defined as one that furthers the individual player's strategy, and are determined through a combination of player-action and game-state prediction. The framework is designed to adapt companion behavior to each individual player, and primary importance is given to ensuring that the companion does not jeopardize the player's perceived strategy in any way. We also propose a method of determining where the companion's potential actions should take place that both offloads the majority of the burden onto the framework while still adapting to player strategy and allowing the game developer to make fine-tuned decisions.

We conducted a user study where participants played "Lord of Towers," a game containing companions using our framework, and received promising results: many participants singled out the companion as a strength of the game without prompting, and a majority of participants agreed that the framework would be useful to them as developers. We would like to continue this project to further improve it and remove some technical limitations we encountered with this version.

9.2 Challenges

At the start of the project a serious amount of time had to be invested in separating the original framework from the Lord of Towers code, as time constraints had led to

more coupling. This ate in to development time, pushing back when the meat of the project could be started. The largest challenge was working with Unity; this was our first time working with the platform as well as the language and a steep learning curve needed to be overcome. Furthermore, while Unity does support multi-threading with its "Coroutines," it does not allow for GameObject-specific code (cloning elements in the game, updating objects, etc.) from happening outside of the main thread. As discussed before, this forced us to perform the state-prediction in the same thread, leading to the blackouts that negatively effected our results.

9.3 Summary of Contribution

We provide a proof-of-concept framework that demonstrates that complementary behavior can be automated. We provided a working definition of a "complementary" action customized to each individual player, based on how it affects both their near-future and long-term goals. The framework was designed to allow for flexibility and ease-of-integration, specifically by performing the majority of the computation in the framework and providing parameters for fine-tuning the process. A new dynamic map-regioning system was implemented that allows the framework to make action-location decisions while giving the designer the leeway to make specific decisions within a returned region. We also refactored the original design of the MimicA framework, simplifying the integration process and decoupling the framework from the example game. We used two separate example games to demonstrate this flexibility; "Lord of Towers" is a modified version of MimicA's original game, and "Lord of Caves" is a similar game featuring procedurally-generated maps.

We performed a user study to quantify the quality of the companion's behavior as well as the usefulness of the framework to game developers, with positive overall results. Of the twenty-five participants, nine mentioned the companion as a stand-out

feature of the game before being asked about the companion, where only four mentioned it negatively. Two of these mentioned that they thought the companion was acting how they would expect another human to act, indicating that the companion's actions were believable. Seventeen users mentioned the companion positively in their answers to the free-response and multiple-choice questions, as opposed to eleven that were negative. Nineteen indicated that the framework could be useful, and twenty said that they would consider using it if the blackouts during the state-prediction process were removed. We expect that the results would be improved further if the blackouts were removed; fifteen participants indicated that it would have made their earlier answers more positive, out of twenty responses. Overall, the large majority of responses were positive and indicated a general satisfaction with the companion's behavior and the framework as a whole.

Chapter 10

FUTURE WORK

The largest roadblock towards this system's usefulness is the single-thread limitation imposed by Unity that forces the game-play to halt when a companion uses state-prediction to choose an action. This reduces the playability of the game enough that it would be unacceptable in a commercial video-game, and likely colors the player's impression of the quality of the companion's choices. Before any other future work could reasonably be done, the system should be migrated to an environment that does not present this problem. The game could potentially be hosted on a separate server, allowing for companion decisions to be calculated in parallel to the player's actions.

More interesting extensions of the system become possible when that obstacle is removed. While the system currently only supports one companion at a time, it would be interesting to modify it to allow more. This would be an interesting experiment, hopefully resulting in the emergence of complementary team dynamics in the companions' behavior. Technically, the system allows for any number of companions but the data from which they would draw their decisions would be shared, resulting in duplicate choices. To fully support this modification each companion would require a separate data set with customized game-state vectors; for example, a vector given to one companion may reflect the locations of the other companions and not their own. Without surpassing the multi-threading limitation it would be infeasible to implement this improvement (one companion is bad enough).

The choice of the probabilities for the randomized aspects of the companion's decisions would also potentially benefit from further investigation. The current values are static and were set by hand after trying multiple values and subjectively choosing

the best. It would be possible to allow these values to change dynamically during game-play as the companion learned from its decisions. For example, one of the probabilities controls whether or not the companion will do the player's predicted next action for them or use state-prediction to choose the most valuable action. This value could potentially change depending on the predicted action; each action could be ranked based on the companions previous predictions of future game states and the probability could be adjusted to reflect higher-ranked actions. Alternatively, some method of player feedback could be introduced allowing them to prioritize certain actions over others, and the probability could be modified accordingly.

A similar method could be used to determine the time the companion spends waiting after each action during the state-prediction process. That time is currently set by the developer, and in our case was set to what seemed appropriate for Lord of Towers. It would be interesting to try to set this time dynamically by attempting to measure how long after an action takes place are the effects of it are still felt. The state-prediction process could also be extended to have the companion try sequences of actions rather than one at a time. This would result in a longer but less frequent process, action sequences that go particularly well together (creating a building and then repairing it to full health, for example) could be identified and paired together into a new "single" action.

More detailed analysis could be done if the framework also included a method of recording and saving gameplay for later use. As this framework evolves it could become a useful research tool for other companion-focused projects, and the ability to record the experiences of the participants in the user study would allow for more in-depth conclusions to be drawn. For instance, had this capability been present during this user study for this project we could have analyzed the games played by those that liked the companion and those that didn't. Potentially, this could have led to interesting findings about what types of behavior players tend to enjoy, allowing

future work on the framework to constrain the companion to those actions.

To make the framework and "Lord of Towers" more useful to future researchers, upgrading the artwork and gameplay would be a benefit. Many of the user responses in the study focused on increasing the quality of the gameplay; the responses might have been more focused on the companions if the production quality of the game was better.

Lastly, as the system is designed to be a portable framework nothing within it is tied to the types of games we used to demonstrate and test it on. It would be interesting to port the logic to other, more complicated games, to further evaluate its usefulness.

BIBLIOGRAPHY

[1] Battlefield 1 - official site. https://www.battlefield.com/. (Accessed on 05/28/2017).

[2] Call of duty. https://www.callofduty.com/. (Accessed on 05/28/2017).

[3] The elder scrolls official site — home. https://elderscrolls.bethesda.net/. (Accessed on 05/28/2017).

[4] Fallout — home. https://fallout4.com/. (Accessed on 05/28/2017).

[5] Madden nfl 18 - football video game - ea sports official site. https://www.easports.com/madden-nfl. (Accessed on 05/28/2017).

[6] Mass effect. http://masseffect.bioware.com/agegate/?url=%2F. (Accessed on 05/28/2017).

[7] Nba. http://www.nba.com/videogames/nbalive10_overview.html. (Accessed on 05/28/2017).

[8] A review of star wars: Republic commando. http://www.reaxxion.com/3250/a-review-of-star-wars-republic-commando. (Accessed on 05/28/2017).

[9] Skyrim is disappointing : why do reviewers ignore its problems ? - the elder scrolls v: Skyrim - giant bomb. https://www.giantbomb.com/the-elder-scrolls-v-skyrim/3030-33394/forums/skyrim-is-disappointing-why-do-reviewers-ignore-it-529212/. (Accessed on 05/28/2017).

[10] Star wars: The old republic. http://www.swtor.com/. (Accessed on 05/28/2017).

[11] Unity - game engine. https://unity3d.com/. (Accessed on 05/26/2017).

[12] Valve. http://valvesoftware.com/games/css.html. (Accessed on 05/29/2017).

[13] A. T. Abraham and K. McGee. Ai for dynamic team-mate adaptation in games. In *CIG*, pages 419–426. Citeseer, 2010.

[14] F. Aiolli and C. E. Palazzi. Enhancing artificial intelligence in games by learning the opponents playing style. In *New Frontiers for Entertainment Computing*, pages 1–10. Springer, 2008.

[15] T. Angevine. Mimica: A general framework for self-learning companion ai behavior. 2016.

[16] S. Butler and Y. Demiris. Using a cognitive architecture for opponent target prediction. In *Proceedings of the Third International Symposium on AI & Games*, pages 55–62, 2010.

[17] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[18] A. Dahlbom. An adaptive ai for real-time strategy games. 2004.

[19] M. Etheredge, R. Lopes, and R. Bidarra. A generic method for classification of player behavior. In *Proceedings of the Second AIIDE Workshop on Artificial Intelligence in the Game Design Process, MJ NELSON, AM SMITH, and G. SMITH, Eds. AAAI Press, Palo Alto, CA*. Citeseer, 2013.

[20] M. Floyd and B. Esfandiari. Building learning by observation agents using jloaf. In *Workshop on Case-Based Reasoning for Computer Games: 19th international conference on Case-Based Reasoning,(Figure 1)*, pages 37–41, 2011.

[21] M. W. Floyd. A comparison of case acquisition strategies for learning from observations of state-based experts. 2013.

[22] M. W. Floyd and B. Esfandiari. A case-based reasoning framework for developing agents using learning by observation. In *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, pages 531–538. IEEE, 2011.

[23] Q. Gemine, F. Safadi, R. Fonteneau, and D. Ernst. Imitative learning for real-time strategy games. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 424–429. IEEE, 2012.

[24] J. Gilroy. Hidden agendas and improved ai in civilization 6 - ign. http://www.ign.com/articles/2016/08/03/hidden-agendas-and-improved-ai-in-civilization-6, August 2016. (Accessed on 11/25/2016).

[25] B. Harrison and D. L. Roberts. Using sequential observations to model and predict player behavior. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 91–98. ACM, 2011.

[26] S. Hladky and V. Bulitko. An evaluation of models for predicting opponent positions in first-person shooter video games. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 39–46. IEEE, 2008.

[27] R. Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM, 2005.

[28] G. Knoblich, S. Butterfill, and N. Sebanz. 3 psychological research on joint action: theory and data. *Psychology of Learning and Motivation-Advances in Research and Theory*, 54:59, 2011.

[29] R. Lopes and R. Bidarra. Adaptivity challenges in games and simulations: a survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):85–99, 2011.

[30] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis. Predicting player behavior in tomb raider: Underworld. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 178–185. IEEE, 2010.

[31] K. McGee and A. T. Abraham. Real-time team-mate ai in games: A definition, survey, & critique. In *proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 124–131. ACM, 2010.

[32] M. Mehta, S. Ontanón, T. Amundsen, and A. Ram. Authoring behaviors for games using learning from demonstration. In *Proceedings of the Workshop on Case-Based Reasoning for Computer Games, 8th International Conference on Case-Based Reasoning (ICCBR 2009), L. Lamontagne and PG Calero, Eds. AAAI Press, Menlo Park, California, USA*, pages 107–116, 2009.

[33] C. L. Moriarty and A. J. Gonzalez. Learning human behavior from observation for gaming applications. In *FLAIRS Conference*, 2009.

[34] S. Ontanón, K. Bonnette, P. Mahindrakar, M. A. Gómez-Martín, K. Long, J. Radhakrishnan, R. Shah, and A. Ram. Learning from human demonstrations for real-time case-based planning. 2009.

[35] S. Ontanón, K. Bonnette, P. Mahindrakar, M. A. Gómez-Martín, K. Long, J. Radhakrishnan, R. Shah, and A. Ram. Learning from human demonstrations for real-time case-based planning. 2009.

[36] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[37] I. Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[38] K. Salen and E. Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2004.

[39] B. S. Sartori L. Complementary actions. 2015.

[40] J. Schaeffer and H. J. Van den Herik. Games, computers, and artificial intelligence. *Artificial Intelligence*, 134(1-2):1–7, 2002.

[41] I. B. W. Theodore Millon, Melvin J. Lerner. *Handbook of Psychology*, volume 5. John Wiley & Sons, Inc., 2003.

[42] M. Tranmer and M. Elliot. Multiple linear regression. *The Cathie Marsh Centre for Census and Survey Research (CCSR)*, 2008.

[43] J. Tremblay. Improving behaviour and decision making for companions in modern digital games. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.

[44] B. G. Weber and M. Mateas. A data mining approach to strategy prediction. In *2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147. IEEE, 2009.

[45] S. Yildirim and S. B. Stene. A survey on the need and use of ai in game agents. In *Proceedings of the 2008 Spring simulation multiconference*, pages 124–131. Society for Computer Simulation International, 2008.

APPENDICES

Appendix A

FEEDBACK SURVEY

This section includes the survey that was given to the participants of the study.

Informed Consent Form

You are invited to participate in a research study entitled "Video-Game Framework Evaluation." The purpose of this study is to investigate a development tool to assist game designers with creating AI controlled companion characters.

This research project is being conducted by the following investigators:
* Gavin Scott, MS Student, Cal Poly
* Foaad Khosmood, Assistant Professor of Computer Science, Cal Poly

Activity: You are being asked to play a game related to the study and then provide feedback through an anonymous survey. The surveys will ask limited questions about your background and opinions related to the game, and to the subject matter. Participation in the surveys will likely involve between 10-15 minutes. None of the activities are strenuous; indeed, they are intended to be engaging and fun. Nevertheless, you may withdraw at any point in the survey without penalty.

Location: The activity will occur online.
Cost: There is no cost to participate in this study.
Compensation: No compensation will be provided for your participation.

Voluntary Nature of Study: Your participation in this study is strictly voluntary. Your grades will not be affected by your participation or lack thereof. If you choose to participate, you can still change your mind at any time and withdraw from the study. If you choose not to participate in this study or to withdraw, you will not be penalized in any way or lose any other entitled benefits. You do not have to answer any questions you choose not to answer.

Potential Risks of Discomforts: There are no risks anticipated with your participation in this study. Only limited identifying data will be collected during the study, so even in the unlikely event of data mismanagement (i.e. unintended disclosure of study data) there is no clear harm anticipated.

Anticipated Benefits: Anticipated benefits from this study are improvements to video-game development here at Cal Poly and beyond.

Confidentiality & Privacy Act: Any information that is obtained during this study will be kept confidential to the full extent permitted by law. Any controlled materials that carry your name (like this one) will be held in an offline, physically secure archive (access to which is strictly controlled). Research results will use only summary and anonymized data. Quoted responses will only ever be anonymous (i.e. "one student observed..."). You will not be mentioned by name by this study. The results of your participation will be confidential.

Points of Contact: If you have any questions or comments about the research, or have questions about any discomforts that you experience while taking part in this study please contact the Principal Investigator, Foaad Khosmood, foaad@calpoly.edu. If you have concerns regarding the manner in which the study is conducted, you may contact Dr. Michael Black, Chair of the Cal Poly Institutional Review Board, at (805) 756-2894, mblack@calpoly.edu, of Dr. Dean Wendt, Dean of Research, at (805) 756-1508, dwendt@calpoly.edu.

Statement of Consent: If you agree to voluntarily participate in this research project as described, please indicate your agreement by selecting "yes" to the following question, playing the complete game, and completing the online survey. Please print a copy of this document now and retain for your reference.

Online Consent: You may be shown this informed consent form in an online form prior to answering survey questions. You can indicate your acceptance by continuing on to the survey. If you do not agree, we ask that you stop immediately and not further continue with the survey.

Do you agree with the above statement? *

- ○ Yes
- ○ No

Figure A.1: The Informed Consent form given to participants

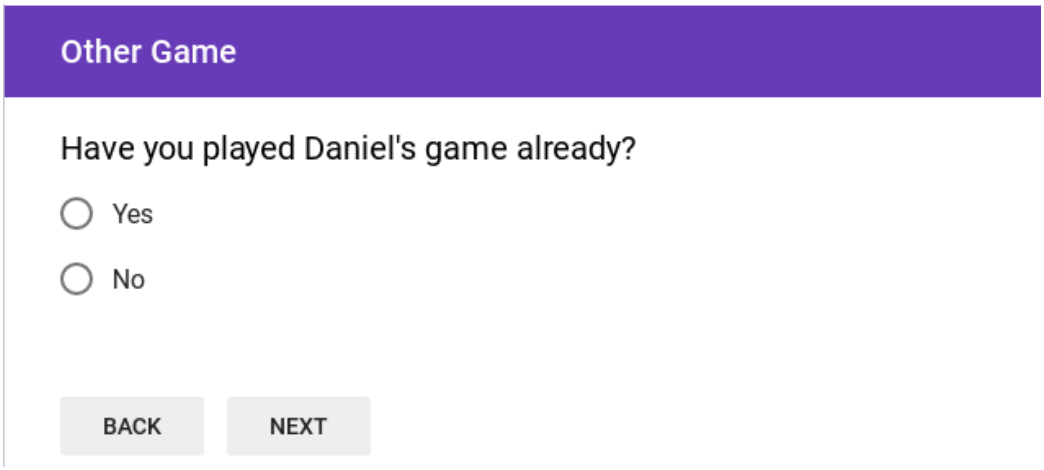

Figure A.2: The first set of questions given to participants

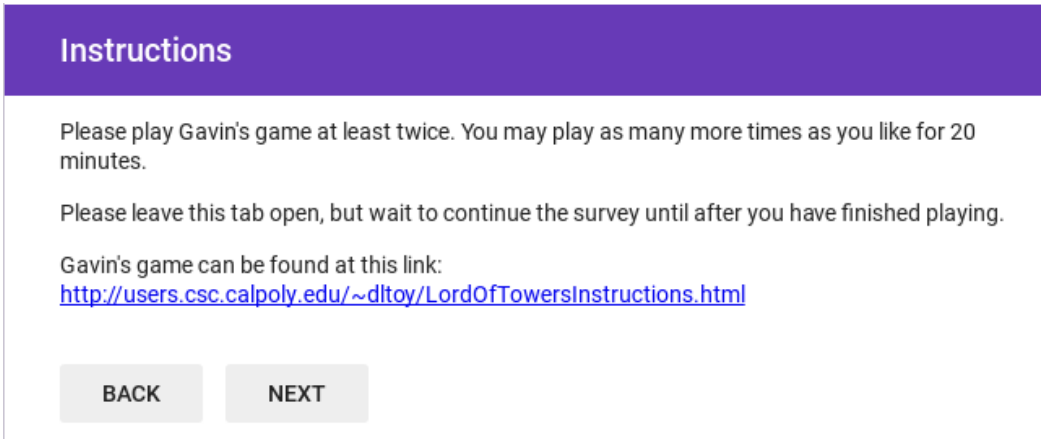

Figure A.3: The second set of questions given to participants

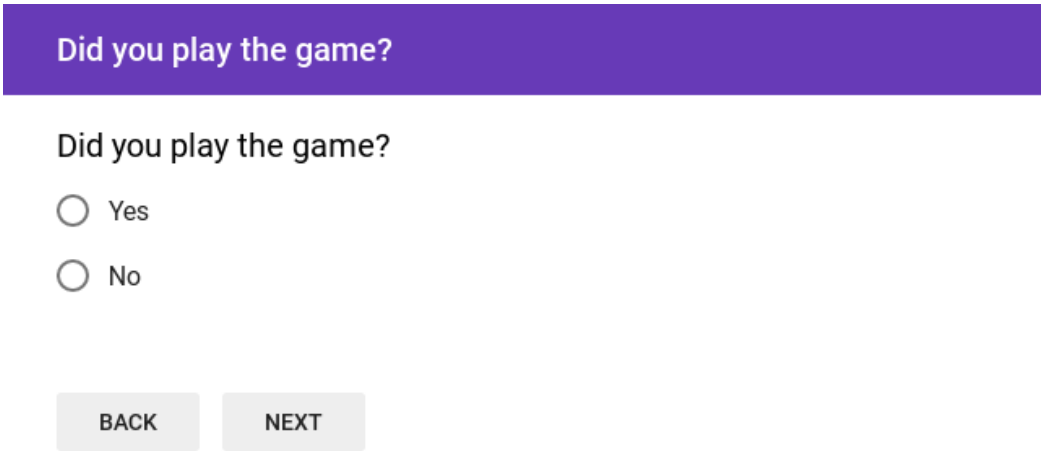

Figure A.4: The third set of questions given to participants

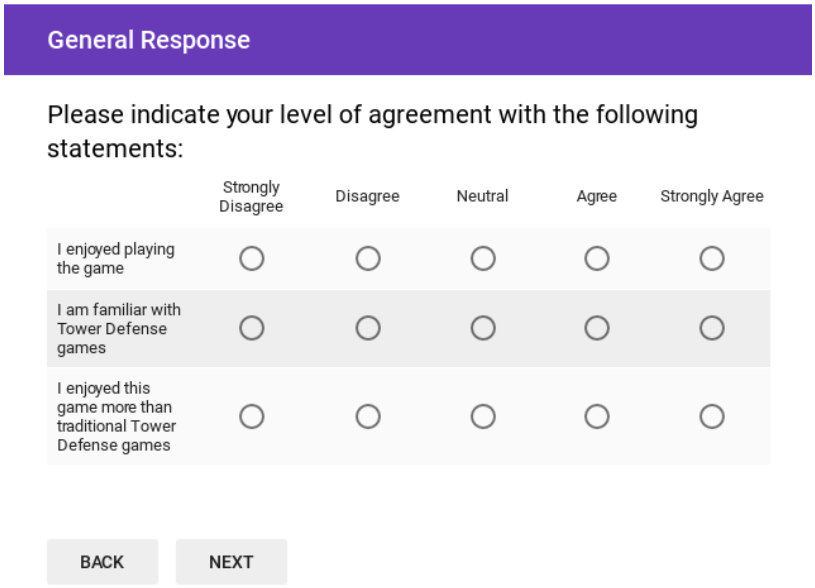

Figure A.5: The fourth set of questions given to participants

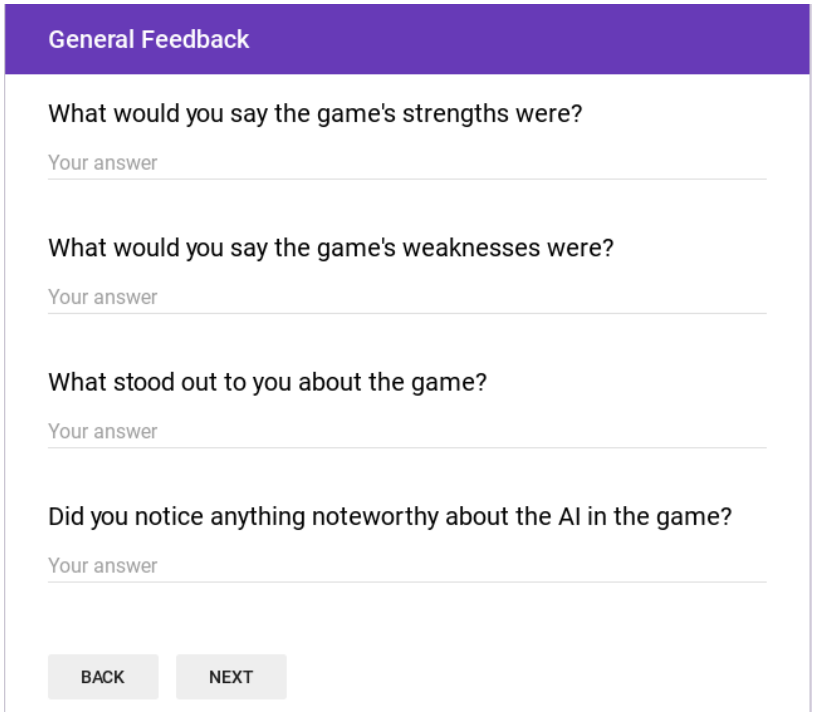

Figure A.6: The fifth set of questions given to participants

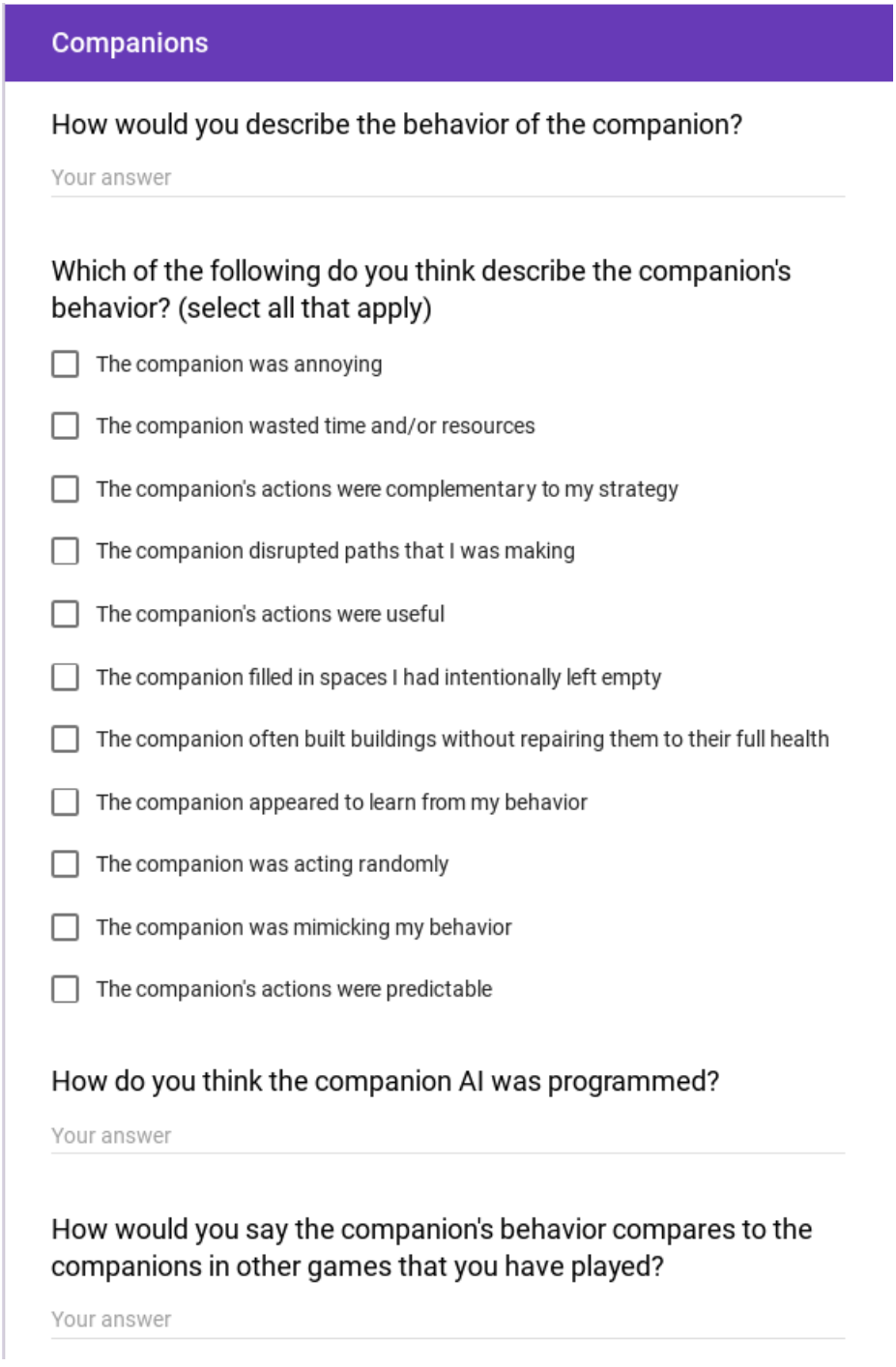

Figure A.7: The sixth set of questions given to participants

Usefulness to Developers

Please read the description of the project, then answer the questions.

Please Read:

PROJECT DESCRIPTION:
The focus of this project was to develop a framework to help game developers add companions to their video games. My project, specifically, was to create a "complementary" companion. Rather than copying the player's behavior the companion attempts to find actions that are complementary to the player's overall strategy. This is done in part by predicting the player's future actions and trying to make sure that the player can do them when they want, trying new actions to explore their effects, and predicting what the game state will be after taking an action to make the best decision at the given time.

THE BLACKOUTS:
The blackouts are a side-effect of the companion's state-prediction strategy. Rather than trying to predict the future state the companion saves the current state, tries and action, then resets and tries the next. During this time the game speed is sped up and the screen is blacked out. This could be done in a separate thread to remove the blackouts altogether, but we were unable to implement that in Unity. In the future, we would like to port this to an engine that supports this process. Please consider the blackouts a temporary feature that could be removed in future versions of the framework.

Please indicate your level of agreement with the following statements:

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
This framework could be useful to a game developer	○	○	○	○	○
I would consider using this framework if I were designing a game with a player companion	○	○	○	○	○
I would consider using this framework if the blackouts were not necessary	○	○	○	○	○

Figure A.8: The seventh set of questions given to participants

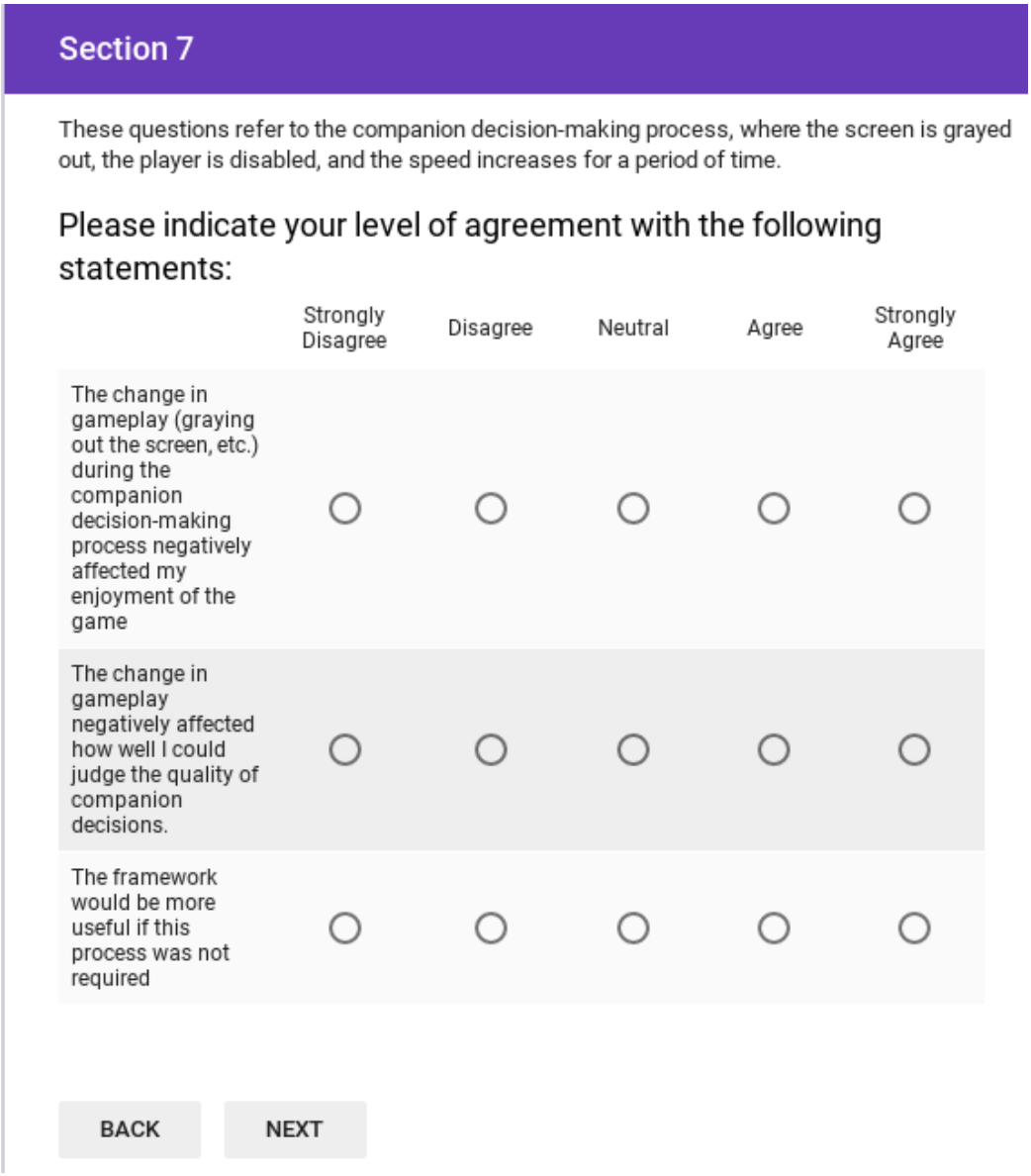

Figure A.9: The eighth set of questions given to participants

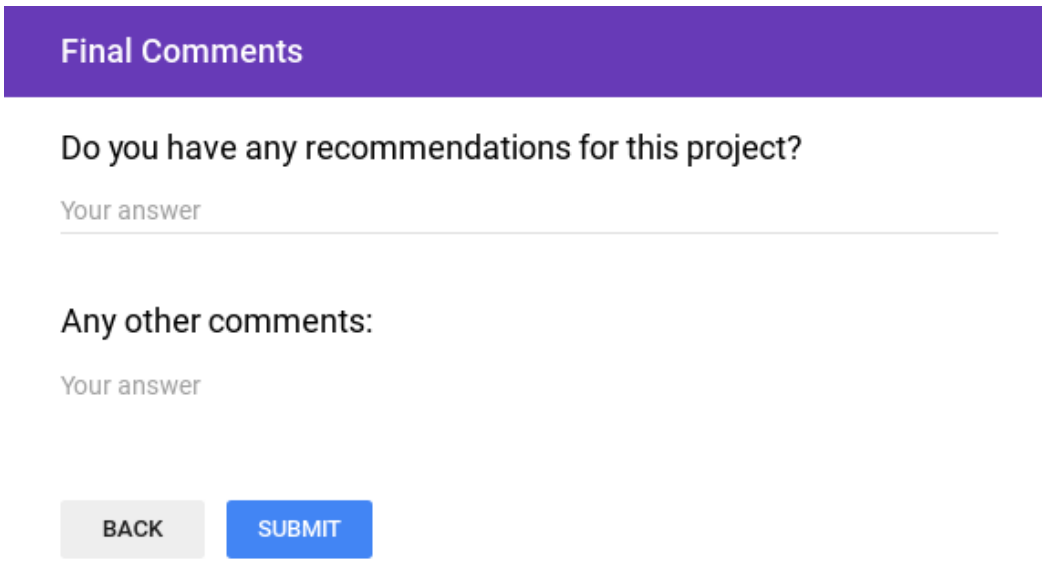

Figure A.10: The final set of questions given to participants

Appendix B

FULL DECISION FLOWCHART

The following page contains the combined flowchart that visualizes the companion's full decision making process.

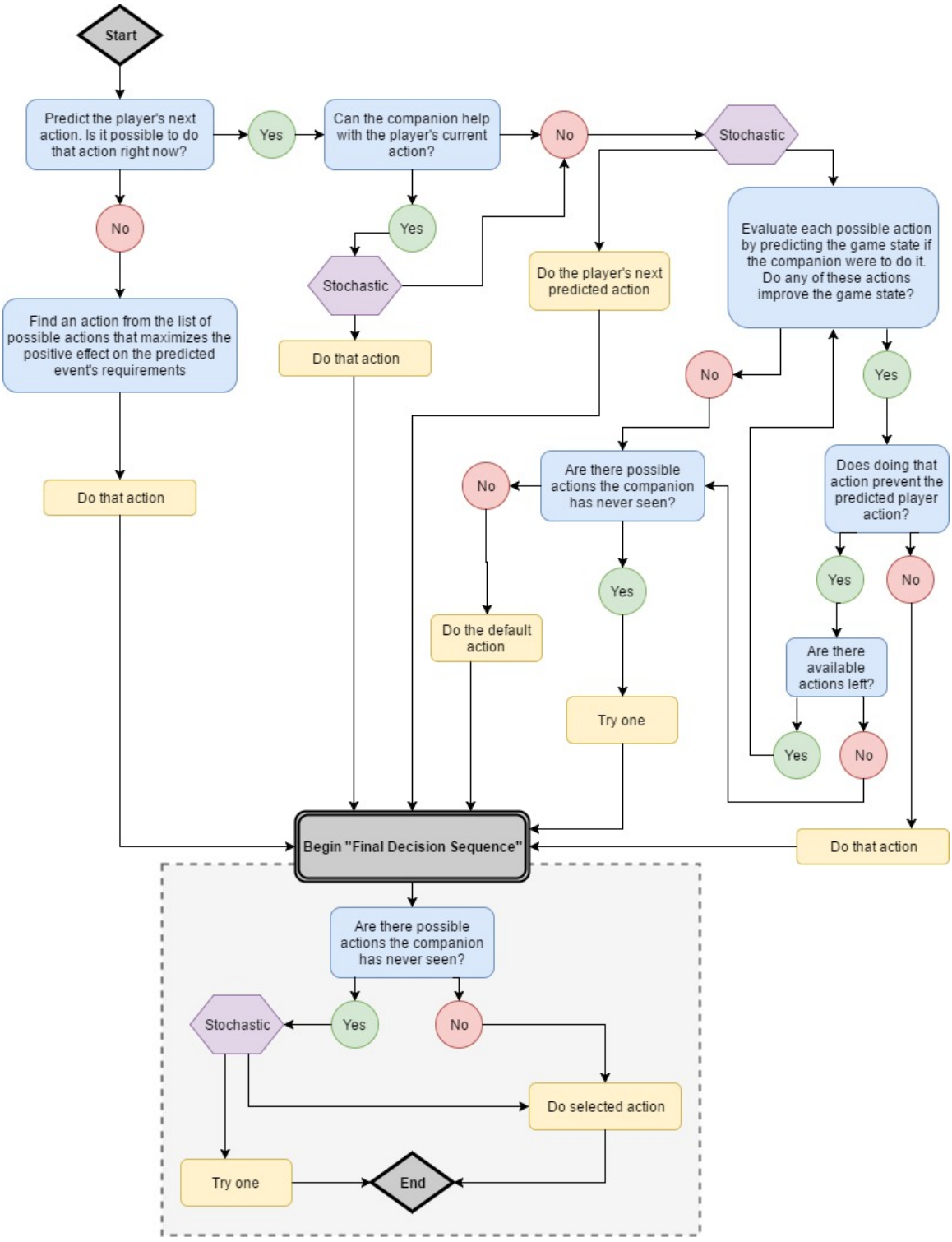

Figure B.1: A flowchart representing the full decision-making process

Appendix C

LORD OF TOWERS ARTWORK CREDITS

C.1 Sprites

- Archer: http://www.sprites-unlimited.com/game/?code=HMM2
- Bomb: https://www.spriters-resource.com/genesis_32x_scd/kidchameleon/sheet/51148/
- Golem: http://opengameart.org/content/golem-animations
- Knight (Companion): http://www.gameart2d.com/the-knight-free-sprites.html
- Mage: http://spritedatabase.net/file/14024
- Dwarf (Player): http://www.sprites-unlimited.com/game/?code=HMM2
- Skeleton: http://opengameart.org/content/skeleton-animations
- Default Tower: taken from Mimica's Lord of Towers
- AOE Tower: http://opengameart.org/content/brick-tower-large-sprite
- Damage Tower: http://ayene-chan.deviantart.com/art/RPG-Maker-VX-Tower-382044826
- Slow Tower: http://cityville.wikia.com/wiki/File:Wizard_Tower-SW.png
- Range Tower: http://opengameart.org/content/cannon-tower
- Vampire: http://opengameart.org/content/vampire-animations
- Zombie: http://opengameart.org/content/zombie-animations
- Ball: taken from Mimica's Lord of Towers
- Castle: taken from Mimica's Lord of Towers

- Cave Border: https://www.pinterest.com/pin/439312138627253428/
- Cave Wall: http://allacrost.org/staff/user/bigpapan0z/ss_browncave.png
- Blue Tile: http://www.iconsdb.com/caribbean-blue-icons/square-icon.html
- Gold Mine: http://ageofempiresonline.wikia.com/wiki/File:GoldMine.png
- Green Grass: https://www.pinterest.com/lcvick/atmospheric-textures/
- Health Center: https://www.reddit.com/r/pokemon/comments/1px3bb/the_advancement_of_the_pokemon_center/
- House: http://www.deviantart.com/morelikethis/297406876
- Log: taken from Mimica's Lord of Towers
- Stone Quarry: http://opengameart.org/content/2d-platform-ground-stone-tiles
- Wall: taken from Mimica's Lord of Towers
- Tree: Ms. Jensen Welton, Artist Extrordinaire

C.2 Sounds

- Ball bouncing: https://www.youtube.com/watch?v=qCm-hjMWnFg
- Basketball bounce: https://www.youtube.com/watch?v=aYvjZSYmkT8
- Bats: https://www.youtube.com/watch?v=nM0InF4UNqU
- Bow String: https://www.youtube.com/watch?v=KW8cSQ3nUj4
- Collapsing Bones: https://www.youtube.com/watch?v=qU_9lnQvLhk
- Death Scream: https://www.youtube.com/watch?v=AGce8M-MZxs
- Evaporating Water: https://www.youtube.com/watch?v=-IqKCSPH-SE&t=123s
- Evil Laugh: https://www.youtube.com/watch?v=ywtjxen2n1A
- Explosion: https://www.youtube.com/watch?v=nRwM7UEQ8Q0

- Fire Sound: https://www.youtube.com/watch?v=vOtzPWx7HXU
- Hammer hit: https://www.youtube.com/watch?v=xcDVBwaI-V0
- Hammer Sound: https://www.youtube.com/watch?v=y5Mw0O0BdwU
- Icicles: https://www.youtube.com/watch?v=Vi6Q86r2UPQ
- Man Scream: https://www.youtube.com/watch?v=H3vSRzkG82U
- Monster Growl: https://www.youtube.com/watch?v=Ii5MaHYlFzw
- Ouch Sound: https://www.youtube.com/watch?v=ZG32UnCzhqE&list=PL6i13PMXG_TezMEGVz6KD_UVINI0FN_sf&index=
- Sizzling Sound: https://www.youtube.com/watch?v=TVyth8uu-w4&t=2s
- Sword Slash: https://www.youtube.com/watch?v=X3liPsg21Cg
- Wand Sounds: https://www.youtube.com/watch?v=FmEiTpMCur8
- Yoga Ball: https://www.youtube.com/watch?v=xeifyIoD6RU
- Zombie Dying: https://www.youtube.com/watch?v=BjirvbYuq7c
- The Dragon Valley by Peter Crowley: https://www.youtube.com/watch?v=HSsO_9DbtOM
- The Kingdom Above the Sky by Peter Crowley: https://www.youtube.com/watch?v=dDwYzJBtv9w
- Peter Crowley's Youtube Channel: https://www.youtube.com/user/PeterCrowley83

Appendix D

INTEGRATION INSTRUCTIONS

This section contains detailed instructions on how to integrate this framework into a new or existing game. This includes descriptions of interfaces and abstract classes that must be extended, and how to fine-tune the framework parameters to best fit an individual game.

D.1 Recording Game Data

The framework requires data to be gathered from the player and statically stored where it can be used during the decision-making process. To facilitate the data storage, the "FrameworkGameData" class is included in the framework. It is also one of Unity's GameObjects, allowing it to be placed in a scene. For it to be used as a data repository a single instance should be added to the scene with the name "DATA", and the addGameStateVector() method should be used to record the player's actions whenever they are taken. Recorded actions must come with the current game state as well as the global coordinates of the action. The player must complete at least one action before a companion can be introduced, but the more actions recorded the better the companion's performance will be.

Game states must extend the abstract FrameworkGameStateVector class, which requires a method to be implemented that returns a numerical score representing the desirability of the game state. A higher score will be considered better than a lower one during the state-prediction process. The game state vector should also include any information required to encapsulate the game state, stored as public

global variables that will be accessed using reflection. These variables could include factors like the player's health and location, the number of enemies on the screen, or anything else that is important to the game. These parameters should be fine-tuned by the developer to give the best results.

D.2 Defining Possible Actions

An abstract class "FrameworkEvent" is included in the framework; each action that the companion or player can take must extend this class. This class requires each event to have a unique name and a method for creating a duplicate event. By default, the location of the event is used during the decision making process, but this can be turned off by setting the "trainOnLocation" variable to "false." It also requires a isPossible() method, which takes in a FrameworkGameStateVector and a FrameworkRegion and returns whether or not the action is possible in that region with the conditions specified in the state vector.

A default action must also be specified (in the FrameworkCompanionLogic class), and a "wait" event is required for the state simulation process and for inserting delays between actions. The duration of the wait action is set to the developer, and should be as short as possible while still allowing for enough time for the effects of an action to be set.

D.3 Enabling Saving & Loading

The state-prediction process requires the companion to simulate multiple actions to judge their outcomes. The game must be saved before this begins, resetting it after each attempted event. This was implemented using the Resettable interface, which must be extended by every object(the player, the companion, enemies, buildings,

etc.) in the scene that must be reset. The interface is simple; the most important methods are saveState() and resetState() which are called during the simulation process on any objects in the scene that extend the interface.

D.4 Retrieving Complementary Actions

Once the framework has been integrated it can be used to get actions for the companion to perform. To use it, create an instance of a "FrameworkCompanionLogic" with your desired parameters. The probabilities for all stochastic chances can be specified, along with a classifier and a method of extracting features from game state vectors if only a subset is required. Am abstract classifier class is also included if custom classifiers are required.

Decisions can be retrieved by the getDecision() method, which requires the players current action and the current game state vector. The first time this is called the classifier will be trained; retraining will only occur when an optional "retrain" flag is passed in to getDecision(). The method will return a "FrameworkEventsToDo" object representing a sequence of actions. In its current form, this is always the complementary action and a "wait" event. Each element in the FrameworkEventsToDo object contains a FrameworkEvent and a FrameworkRegion representing where it should take place. It is up to the developer to determine where exactly in that region to perform the event. Regions are rectangular, and represented by their bottom left and top right corners in global coordinates.

Appendix E

DECSION-MAKING ALGORITHM

Algorithm 1: Companion's Decision-Making Process

Data: `curState`, `playerCurAction`, `unseenActions`
Result: Companion's chosen action for the current game state

```
// predict player's next action;
predicted = findPredictedAction(curState);
if not predicted.isPossible(curState) then
    // find action that makes predicted action possible;
    chosen = doStatePrediction(curState, predicted);
else
    if playerCurAction.allowsMultipleActors() and chance() then
        chosen = playerCurAction;
    else
        if chance() then
            chosen = predicted;
        else
            // get list of possible actions, ordered by effect on the state;
            // actions that decrease the state score are not included;
            actionList, resultStates = doStatePrediction(curState);
            ndx = 0;
            // find best action that doesn't prevent predicted action;
            while not actionList.isEmpty() and ndx <actionList.length do
                if predicted.isPossible(resultStates[actionList[ndx]] then
                    chosen = actionList[ndx];
                    break;
                else
                    ++ndx;
                end
            end
            // no valid actions in actionList;
            if unseenActions.length >0 then
                chosen = chooseRandom(unseenActions);
            else
                chosen = defaultAction;
            end
        end
    end
end
// potentially choose unseen action;
if unseenActions.length >0 and chance() then
    return = chooseRandom(unseenActions);
else
    return chosen;
end
```

Figure E.1: An algorithm describing the complementary decision-making process