Player Analytics: Papers from the AIIDE Workshop
AAAI Technical Report WS-16-23

MimicA: A General Framework for Self-Learning Companion AI Behavior

Travis Angevine and **Foaad Khosmood**
Department of Computer Science and Software Engineering
California Polytechnic State University
San Luis Obispo, California, USA

Abstract

We explore fully autonomous companion characters within the context of Real Time Strategy games. Non-player Characters that are controlled by Artificial Intelligence to some degree, have been a feature of Role Playing games for decades. But RTS games rarely have a player avatar, and thus no real companions. The universe of RTS games where both an avatar and a companion character exist is small. Most friendly RTS units are semi-autonomous at best, requiring player micromanagement of their behavior. We present MimicA, a real-time framework to govern AI companion behavior by modeling that of the current player. Built for the Unity engine, MimicA is a learn-by-demonstration framework that differs from existing practices in that the behavior is fully autonomous, does not rely on previous modeling exercises and is designed to be generalized and extensible. We analyze and discuss MimicA through a thirty-person user study with our own demonstration game, *Lord of Towers*. We find that 22 out of 30 participants (73%) indicate they enjoyed the game, and this self-reported enjoyment was on par with "traditional tower defense games". 63% agree that MimicA controlled NPCs are doing what the player would do while 20% disagree. Similarly, 53% realize the NPCs are learning from the player while 20% do not. We also show that NPC with underlying Decision Tree and Naive Bayes algorithms are better than KNN in making the player realize the learning nature of the NPC.

Introduction

As video games have developed from the early days of *Pong* (Atari Incorporated 1972) and *Tetris* (Sega 1989) to 21st century hits like *World of Warcraft* (Blizzard Entertainment 2004) and *Call of Duty: Modern Warfare* (Infinity Ward 2007), they have evolved in their style, depth, and difficulty. The range and depth of artificial intelligence (AI) techniques used by the non-player characters (NPCs) in the games, have grown as well. AI-controlled characters include enemies that oppose the player, functional characters that may support the player indirectly in shops or quests, and companion characters that work alongside the character, often for the purpose of amplifying or complementing the player avatar's abilities.

Previous research has focused on making these types of characters believable (i.e. more human) (Livingstone 2006), and to make them player-like (i.e. more complex with varied behaviors and responses). While much work has gone into developing highly sophisticated AI for adversarial characters, less has been done for companion characters (Bakkes, Spronck, and Postma 2005)(McGee and Abraham 2010), and almost none for companions within Real Time Strategy (RTS) games. Major contemporary trends in companion AI development are toward either creating fully autonomous companions, or creating companions that are still controlled by the player to some degree (Tremblay and Verbrugge 2013).

This work falls into the first category by focusing on developing characters that behave completely without explicit player direction.

Good AI companions will aid in increasing the fun and immersion of a game (Tremblay and Verbrugge 2013), as well as providing more believable player-NPC interactions. They could allow for more complex strategies to be used both by game developers and the players because the NPCs working with the players will be closer in level of competence to the current state of enemy NPCs. Additionally, smarter AI companions could avoid the problems mentioned previously in the reviews of Skyrim companions. MimicA aims to provide better experiences through the creation of fully autonomous companions based on a model of the main character.

To evaluate MimicA, we must first establish that it can produce NPC agents that achieve the goal of imitating a particular player. Second, we want to make sure the NPC is actually useful and not a hindrance to game enjoyment. Third, we'd like to compare and contrast the three MimicA-supported machine learning algorithms. Evaluations are based on a thirty person survey where each subject plays the game with one of three algorithms and fills out an online survey after their session.

Related Work

Several important lines of research in player modeling serve as guides and background for this paper. (Van Hoorn et al. 2009) examine direct and indirect player modeling in context of auto racing games. Using machine learning, the authors create autonomous vehicle controllers with training

data provided either entirely based on human play sessions, or one evolved with the help of additional domain-dependent metrics specified by the authors. A similar dichotomy is pursued in (Holmgård et al. 2014) where "model-free" player "clones" are compared with model-based "personas" generated with the help of some domain knowledge. (Pedersen, Togelius, and Yannakakis 2009) generate parametrized Super Mario levels and correlate them to player feedback regarding feelings such as fun, frustration and challenge.

Two systems provide related work to MimicA: jLOAF and Darmok 2.

The java learning by observation framework (jLOAF) is a similar framework presented in (Floyd and Esfandiari 2011b) and followed up in (Floyd and Esfandiari 2011a). The framework aims to aid in the development of agents in different environments, where the agents learn the behaviors they will perform without explicitly being told about necessary tasks or goals. They use case-based reasoning for action determination, and the framework breaks actions and inputs into atomic and complex parts in order to better represent possible inputs to the system and actions to perform.

Darmok 2 (D2) is a real-time case based planning system for RTS games developed by (Ontanón et al. 2009). D2 is a planning system designed to be domain independent, capable of learning how to play RTS games through human demonstration. D2 uses demonstrations, plans, and cases in order to operate effectively. Demonstrations in D2 are represented as time, state, action triples. Since actions in RTS games are not always successful, D2 adds more than just preconditions and post-conditions to the actions, including success conditions, failure conditions, and pre-failure conditions. Demonstrations can then be combined into plans consisting of transitions and states. These plans are then stored as cases. Cases also contain episodes, which is an object containing the outcome of a plan when executed at a specific game state. In addition to human demonstrations, D2 requires a set of goals, set beforehand on a per domain basis, that it looks for in the plans that are obtained from the demonstrations. After D2 has a case base which it will operate off of, when it retrieves a plan from the case base it attempts to modify the plan to fit the specific situation before acting on that plan.

We aim for a system with total autonomy which (Togelius, De Nardi, and Lucas 2007) call "imitation", and (Holmgård et al. 2014) calls "clones". In addition, we aim for a generalizable system with no expert goals defined, and with focus entirely on the current player. Thus we do not consider using training data from other individuals or data collected offline.

The MimicA System

MimicA is a two part system consisting of action observation and action determination. Each part is outlined in this section, followed by a brief discussion of how a game developer would interface with our framework. Figure 1 shows a general flow diagram for how the MimicA system works, in order to provide reference for the steps outlined in this section.

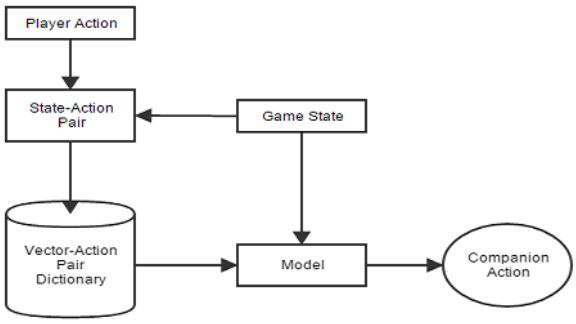

Figure 1: A flow diagram for MimicA

Model Building

MimicA is built to interact with a game through observation of actions performed by the player in the current game session. These can be any move, or possible lack of move, that a player of the game could make through the normal course of gameplay using intended interfaces. Specific state information about the action, such as where it was triggered, is maintained in order to provide the system with context. However, details about the exact object (target) that the action was performed on are not maintained for two reasons.

First, the exact object may change in the game. For example, an attack action could be performed; however storing the specific enemy that was attacked is not useful because that exact enemy may not exist the next time an attack action needs to be performed. Secondly, we want the system to be as generic as possible. It should determine through game play what needs to be done and where. So using the attack example, while the same enemy might still be in the game when the AI determines what to do next, it may be better for the AI to attack a different enemy, based on the current state of the game.

Any time an action is performed by the player it is paired with the state of the game at that moment in time, and recorded by the system. The game state is represented by a vector of features that are designed to capture any and all important aspects of the game at any given point. The game designer, through an interface with the MimicA library, provides this game state, or feature vector. It is left up to the designer to decide what features to include. Once the feature vector has been created, and it has been paired with the action that was just performed, this vector-action pair is stored in memory for later processing.

Action Determination

When the AI companion needs to determine what action to perform next, it once again creates a vector for the current game state. MimicA offers three different supervised classifiers to determine which action to take based on the created vector: K-Nearest Neighbor, Decision Tree, and Naive Bayes.

API

MimicA, in its current state, is developed in C# for the Unity game engine. A basic class diagram can be seen in figure

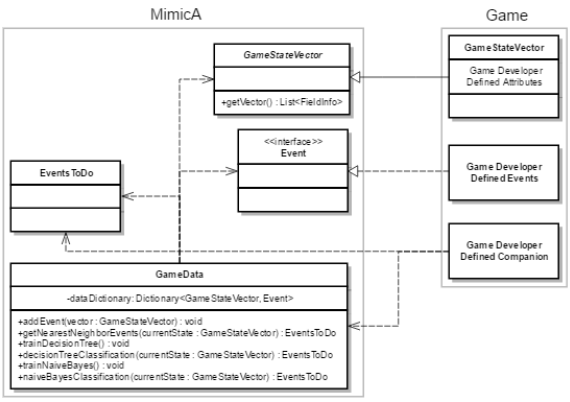

Figure 2: A class diagram for MimicA and its basic interaction with a game that uses it

2. The game developer is required to extend the GameStateVector class and implement the Event interface. The game developer's GameStateVector class will contain all of the state information, through private instance variables, that are important to determining what action to perform next. They are then able to interface with MimicA through the "GameData" class in order to add new state-action pairs to the dictionary, train the classifier if necessary, and request the next action to perform.

MimicA in Lord of Towers

As part of this project we develop a tower defense game, *Lord of Towers*, to go along with MimicA and aid in validating the features of the system. A few notable things about the game are that the player has a physical presence in the game, there is no pre-defined path for the enemies, and after a certain amount of time, the player character will always die, but the game is meant to continue with the companion character. The player is meant to become an observer at that point until the true end of the game. Like other tower defense games, a critical number of "breaches" of enemies reaching the base constitutes failure and ends the game.

The basic back-story is that a village is under attack by hordes of aliens. The player is the only person with skills to construct towers and fight back against the enemies. However, the player's character is already injured and will be dead soon. The only hope is for the player to hold out long enough until the village can produce more champions who are being trained by watching the player. Once the player is dead, the other champions will continue to resist.

At the beginning, only the player is available to defend, as in normal tower defense games. After a preset duration of time, the first companion is introduced and will proceed to assist the player by performing any of the tasks that the player has performed previously. Second and third companions will join the game in order with passage of time. These companions operate based on the same stored player-derived data as the first companion, however all of them are able to move independently. After a preset timer expires, the player is removed from play and the companions are left to fend for themselves. At that point, MimicA stops recording state/action pairs, and the model building part of the system is completed.

Game State Representation

The game state in *Lord of Towers* is represented by a vector of about 250 values that update within the game loop. One of the challenges of the design was how to represent spacial references within the vector. Our solution includes both relative and absolute positioning. The 2D game space is divided into six rectangular sectors, and statistics from each sector are included in the vector. Separately, a set of features describing the sector the player avatar was currently in is also included. Some relative distances such as distance to "nearest tower" and "nearest enemy" are recorded as well. Below we outline the different sets of values used to construct the feature vector.

- General Game State
  - Current avatar health
  - Number of enemies currently in combat with the avatar
  - Current score
  - Resources available
  - Total number of enemies of different types currently on screen
  - Distance of the closest enemy to the base
  - Distance of the closest enemy to player avatar
  - Number of companions currently alive
- Towers and Walls
  - A histogram of the last N readings of health (hit points) of all towers by sector (a typical number for N is 5)
  - Health of all walls by sector
  - List of sectors where at least one tower is under attack
  - List of sectors where at least one wall is under attack
- Previous Actions
  - Previous N actions performed
  - Time elapsed since each of the N actions were performed

The game models resources. Resources are spent on building creation and repair. They are earned based on enemy destruction. The resources are shared amongst the player and all the companions.

The game allows the player to do four general actions: build a new building (tower, wall, medical center), repair a building, fight an enemy and stand guard. Build and repair are closely related because the build move is actually implemented as "initiate" followed immediately by the repair action that actually completes the building. "fight" and "stand guard" are also very similar because the companion is programmed to attack an enemy that is close to it if no other orders are currently being carried out. Thus to "fight" is to send the companion near the enemy. Build actions also have a few built-in assumptions in this game. There is a preset minimum distance between towers. When building walls, the companion generally recognizes existing horizontal and

vertical arrangement patterns and attempts to continue them. All actions are associated with a sector number in which to carry out the action. Available build actions are filtered by the ones that are currently affordable.

User Study and Results

In order to test the effectiveness of the MimicA framework we ask subjects to play *Lord of Towers* and answer a survey about their experience[1]. A general call is made in the university related social media for subjects.

We evenly test each of the three classification methods in order to determine if one of the methods is perceived to produce better companion behavior. Each participants is randomly sent one version of the game. The participants are told nothing about the companion other than it will help them in the game.

After participants finish playing the game they are asked to take a web survey about their experience. The survey includes questions about both the game and the AI companion, and includes both free-form responses and multiple choice questions. The entire process takes about 15-20 minutes on average. Subjects can win one of several $20 steam cards as a reward for participation and thus only provide an email, confidentially kept for that purpose.

As a part of the analysis of the participant responses, we code one of the free-form answers that we receive. The question is "How do you think the companions were programmed?" Crucially, this question is asked at a point in the survey where no information or hints about the real behavior of the NPC companions are given to the user, in order to minimize bias.

We ask three coders, individuals familiar with the project, to independently code the responses with one of four possible categories, agreed upon ahead of time. These categories, as well as a sample response that fell into each category can be seen in table 1. As long as two of the three coders agree on a label for a particular response, we count that as a true response in that category. A majority exists in all 30 cases. The results for 18 out of the 30 responses are unanimously coded. The other 12 responses are coded based on majority (2 out of 3).

The first few questions of the survey are intended to illicit feedback about how players felt about the game itself. We ask users to respond to two statements with how much they agree statements "I enjoyed the game" and "I enjoyed the game more than a traditional tower defense game". The possible answers are those of the five-point Likert scale ranging from "strongly disagree" to "strongly agree". The results are shown in Figure 3. As the graph shows, a 22 out of 30 (73%) agree or strongly agree with "I enjoyed the game". However, on average, participants are neutral about enjoying the game more than a traditional tower defense game. Determining if the participants found the game enjoyable is important to make sure that our system does not hinder player enjoyment of the game.

Next, the survey focuses on companions in the game. First, we ask participants, "How do you think the companions are programmed?" This is a free-form response question, the answers of which are coded into categories found in table 1. The results of this coding can be seen in Figure 4. Nine of the 30 responses are coded with "2" and "3" indicating the behavior does not depend on the player, and does depend on the player respectively. An equal number of the responses could not be categorized, usually with answers like "I don't know".

The next question is a multiple choice (select all that apply) question about the behavior of the companion, asking participants to indicate if they noticed the companion doing any of the descriptions given. The possible options, as well as the responses, can be sees in Figure 5. 22 of the 30 participants (73%) indicate that they notice the companion performing "similar actions" to themselves. Additionally, 17 of the participants feel the companions are performing actions that are useful to them.

Next, participants are asked to rate their agreement with a number of statements, this time focusing on the companion. The statements are "The companion/s was/were useful to me", "The companion/s would protect me", "The companion/s was/were performing actions that I would do", and "The companion/s was/were learning from the actions that

Code	Code Category	Sample Response
1	They built things regardless of what else was going on	They appear to move and build at random
2	They do what is needed based on what else is going on, but do not rely on player behavior	Finite state machines
3	They mimic the player or were effected by player behavior in some way	To replicate what the user is/has been doing
4	Other	No idea

Table 1: Coded categories and a corresponding sample response

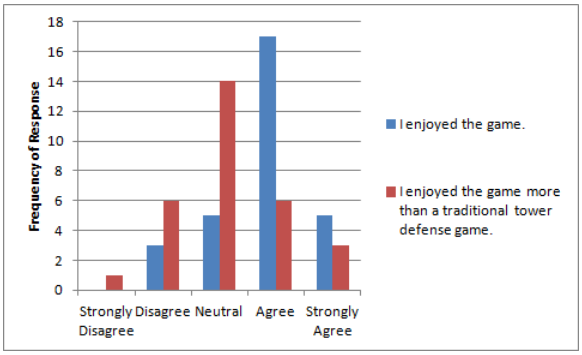

Figure 3: Participant responses with regards to how much they enjoyed the game

[1]Game and survey can be found here: http://goo.gl/Q0NyU2.

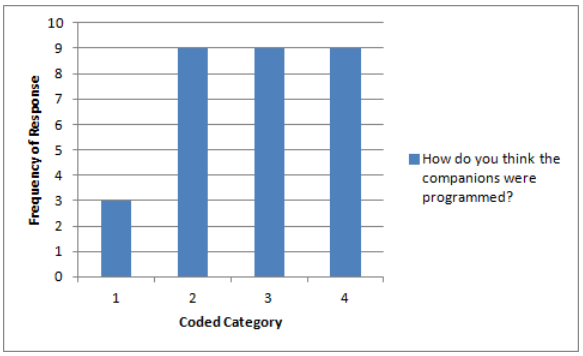

Figure 4: Coded responses for free-form question "How do you think the companions are programmed?"

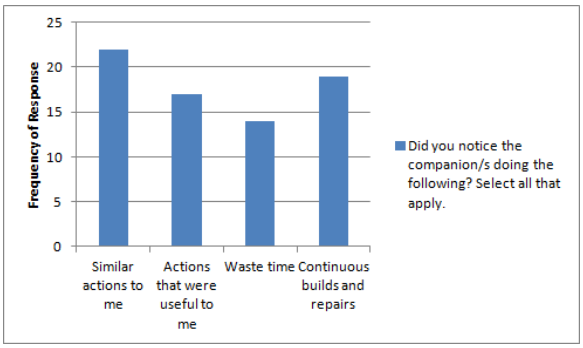

Figure 5: Participant responses when directly asked about various companion behavior

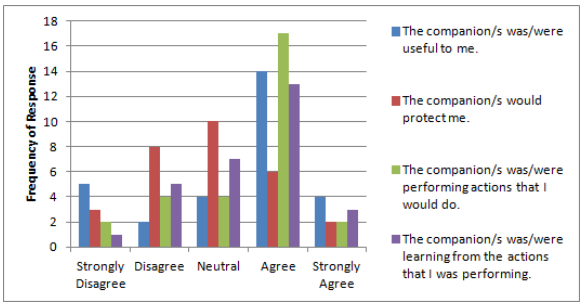

Figure 6: Participant responses, on a Likert scale, regarding companion behavior

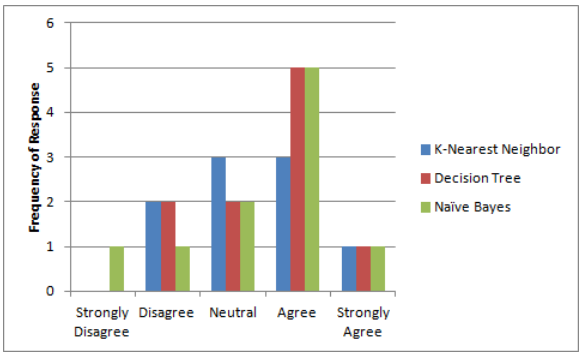

Figure 7: Participant responses, separated by classification method, for agreement on the companion learning from actions they were performing

I was performing", answering on a Likert scale. While two of these statements aim to gather much the same data as is discussed above and presented in figure 5, the final statement is the most important of the group. We are directly asking participants if they notice any form of learning behavior based on the player. The results of this question can be seen in Figure 6. When directly asked, 53% either agree or strongly agree the companions are learning from actions that the player is performing. Only six of the participants (20%) disagree and the rest are neutral.

We separate the results by classification method. As can be seen in Figure 7, the Decision Tree method and the Naive Bayes method have the best response, each having six of the ten participants per method who either agree or strongly agree that the companions are learning from the actions the player is performing. K-Nearest Neighbor only has four participants who either agree or strongly agree, and was on average neutral.

We also ask the participants if they ever wish the companions would do something that they were not, and if they indicate yes we asked what they wished the companions would do. 23 out of 30 of the participants wish the companions would do something that they were not. The majority of the responses to the follow up question, however, seem to indicate that the problem was likely in the game rather than the MimicA framework. Most of the responses were along the lines of "don't disrupt the path that I created", or "don't fill in the gaps that I leave to make a maze", or "companions needed to repair buildings that they just built".

With respect to the last response, it is a known problem in case based learning that, depending on the method used, actions performed by the agent may not be in the same temporal order as those performed by the expert (Floyd and Ontañón 2013). However, this could also be solved in the game. As it stands in *Lord of Towers*, build and repair are two separate actions, leading to the observed problem that companions do not always repair a tower to full health right after they build it. While we do not want to remove the repair action all together, it would make sense from a game development standpoint to immediately follow the build action by a repair action for the companion, just as it works for the player. This would not prevent other companions or the player from helping to "repair" a newly constructed building to full health, but it would result in more fully constructed buildings. Although this is a known issue for case based learning agents, this could likely be solved on the game side, as opposed to relying on a solution on the framework side. Alternatively, if a solution could be found on the part of the framework, it could possibly open up a wider range of dynamic behaviors where the companion decides it does not need to finish building something because there is a more urgent need elsewhere, and instead will come back to finish the building after.

Near the end of the survey, we state on a new survey page that the companions are indeed programmed to learn from the player's behavior, and ask the participants to take the perspective of a game developer to answer the question "If this AI was available as a library/plugin that you could use to aid in development of your game, would you use it?", and why or why not? This question garners mixed results, most likely due to the broad range of participants that are in the study. 21 of the participants say they would use such a plugin, eight say they would not, and one does not provide an answer. It may be important to note, however, that two of the participants who say they would not use it, followed up by saying it is because they are not game developers.

Of the 21 that indicate they would use a plugin like this, many of the follow up responses indicate they would use it because it would take away some of the load for the game developer or that it would help expand upon the possible strategies available in the game, both features that MimicA aims to provide. One of the main points of opposition to a companion like this is that the system does nothing to support a complimentary companion, one that does not strictly mimic the player, but tries to balance out the player's abilities by performing useful actions the player is not doing.

Conclusion and Future Work

In this paper, we present the MimicA framework, a system for governing the behavior of companion AI based on directly learning from the player in the same session. We posit that certain games can benefit greatly from an open framework designed to fully automate the companion AI for those games where it makes sense to have companions learn behavior through actions of the player. The challenge is for the task assignment system to intelligently choose the right companion and assign it the right task at the right time. This study presents three different classification methods, in order to see if one method is perceived to be better than the others as judged by the final impression of the player.

Our evaluation using the game *Lord of Towers* indicates that MimicA-produced NPC are recognizable as agents that do similar actions as the player (63% agree versus 20% disagree), and that they are learning agents (53% agree versus 20% disagree). The overall player enjoyment is 73% with a slight majority even enjoying the game more than a traditional tower defense game (30% agree versus 23% disagree).

When separated by classification method, more participants who use the Decision Tree or Naive Bayes methods agree or strongly agree that the companion learns from the player (6) compared to those who used K-Nearest Neighbor (4). Of the three classification methods, participants who use the Decision Tree method generally have the best response. Users generally understand what companions are doing and find them helpful.

For future work, we would like to integrate MimicA with an already functioning game, or multiple distinct games. While *Lord of Towers* is good as a test bed for the framework, too many of the problems that arose in development or that were brought up in our study could have been the result of the game, not the framework. As such, using the framework with an existing game would be beneficial to clear up some of the possible issues. At the same time we could evaluate the generalization of the system which is largely unaddressed in the present study.

In its current state, MimicA has no form of planning incorporated with the methods in which it determines what action should be performed next. Adding a planning system to the framework would allow MimicA to perform more advanced action determination, thereby enhancing the performance of the framework. A planning system could partly address some of the specific feedback from user study participants who point out the companion NPC failed to see the bigger picture and would often disrupt a larger pattern of tower building by the player. Perhaps with some planning capability, the NPC may be able to recognize more of an overall intent and fit its own work within that effort.

Additionally, it would be beneficial to detach MimicA from Unity. While developing *Lord of Towers* in Unity made the most sense based on time constraints and prior knowledge, it restricts the possible audience for the framework. Being able to separate MimicA away from Unity into just a C# library, or even be able to implement it in other languages, would be highly beneficial towards expanding possible use cases.

References

Atari Incorporated. 1972. Pong. [Arcade Game].

Bakkes, S.; Spronck, P.; and Postma, E. 2005. Best-response learning of team behaviour in quake iii. In *Workshop on Reasoning, Representation, and Learning in Computer Games*, 13–18.

Blizzard Entertainment. 2004. World of warcraft. [PC Computer, Online Game].

Floyd, M., and Esfandiari, B. 2011a. 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)*, 37–41.

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

Floyd, M. W., and Ontañón, S. 2013. A comparison of case acquisition strategies for learning from observations of state-based experts.

Holmgård, C.; Liapis, A.; Togelius, J.; and Yannakakis, G. N. 2014. Personas versus clones for player decision modeling. In *International Conference on Entertainment Computing*, 159–166. Springer.

Infinity Ward. 2007. Call of duty: Modern warfare. [PC Computer, Playstation 3, Xbox 360].

Livingstone, D. 2006. Turing's test and believable ai in games. *Computers in Entertainment (CIE)* 4(1):6.

McGee, K., and Abraham, A. T. 2010. Real-time team-mate ai in games: A definition, survey, & critique. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, 124–131. New York, NY, USA: ACM.

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

Pedersen, C.; Togelius, J.; and Yannakakis, G. N. 2009. Modeling player experience in super mario bros. In *2009 IEEE Symposium on Computational Intelligence and Games*, 132–139. IEEE.

Sega. 1989. Tetris. [Arcade Game].

Togelius, J.; De Nardi, R.; and Lucas, S. M. 2007. Towards automatic personalised content creation for racing games. In *2007 IEEE Symposium on Computational Intelligence and Games*, 252–259. IEEE.

Tremblay, J., and Verbrugge, C. 2013. Adaptive companions in fps games. *FDG* 13:229–236.

Van Hoorn, N.; Togelius, J.; Wierstra, D.; and Schmidhuber, J. 2009. Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation*, 652–659. IEEE.