Front Cover

Disclaimer

Acknowledgements

Abstract

There are currently very few shop management games, and training simulators. Games currently available are very simplistic and hard to navigate. Working as Customer Service Assistant (CSAs) and working as a manager, or supervisor, are very different tasks and sometimes the jump from one to the other is too large. A big problem is the lack of a way to show how managing a shop work and, how to be good at organizing and running a shop.

Creating a realistic, in-depth simulator/game, to show off the shop environment would be a great addition to a shop’s training system, and can be a great way for people to spend their time, while also developing their skills in important real life ways, such as money management, customer satisfaction, team management etc.

The program created during this project will not be a full game, it will be a stripped down, simpler version to simulate and how off the AI systems and algorithms.

Contents Page(s)

Chapter 1: Introduction including aims and objectives.

Introduction:

Working in a shop, of any kind, can be overwhelming at first. There is such a huge range of personalities that cashiers and supervisors need to deal with daily. Usually, it is a learn-as-you-work experience since there is no manual, or cut and paste solution for dealing with difficult customers. The program developed throughout this project will be the baseline for a complex, and AI intensive simulation which can be used by shop employers to help train their employees by manufacturing complex and involved scenarios. The program will provide intelligent AI, which reacts realistically to the employees in the simulation, and a user can control an employee, so that they can try different solutions for dealing with difficult customers. The employees can be taught, and practice dealing with complicated customers and situations without it happening in real life, which means that if they perform badly, the shop’s reputation does not get damaged and customers does not get upset or angry.

 Upon initial research shrouding this type of educational program, it was discovered that there are very few, if not any, exceptional simulations that can be used in the way described above. One 3D version of a shop management game [1] is made for mobile only, and at first glance does not look too advanced or developed. Due to the lack of good research material being available for shop management games specifically, they next best place to take inspiration from would be simulator games with highly intelligent AI, such as RimWorld [2] and Prison Architect [3].

Figure 1: RimWorld pathfinding example

Using management games with intelligent AI as main points of inspiration, this project will be focused upon creating a basic program, with basic user interactions, but with a complex and advanced AI, which will show off the potential educational simulation that will be developed using this program as its baseline. The program will have a top-down/bird’s eye view camera set-up, as this allows the most amount of visibility of the entire shop, and helps the user get the best assessment of the AI systems.

Aims:

The final program that will be developed by the end of this project, will be used as an example, and proof of concept item, of a potential simulation full of complex and realistic AI decisions, which can be manipulated by the user to create scenarios which then can be used to train and teach employees about how to deal with an enormous amount of different shop interactions and scenarios.

This project’s AI systems will include:

* Pathfinding – Research will be done to find the best pathfinding algorithms, which will then be implemented to create a realistic looking pathfinding system.
* Job Decisions – The employees will need to decide which job should be done depending upon criteria such as customers in the store, and the time of day. Research will be done on games already using this system, such as RimWorld [2]
* Shopping Choices – The customers will need to know what products they are after and then look for them. If the customer finds the items they want, they need to decide on whether to purchase then based upon price, quality etc.
* Relationships – The characters in the program, employees and customers, will have different relationship levels. These levels will be used to determine if two characters want to engage in a conversation, and then their relationship will increase of decrease based upon that interactions, and the characters’ traits.

The base program system/ UI implementation needs to be completed so that the user can interact with the world. The user should be able to play the program and get a feel for what a final, fully developed version of this program will be like. The main systems that will be developed and implemented are as follows:

* World/Tile Interactions.
* General Character Pathfinding and Interactions
* Employee Job Decisions
* Customer Decisions

All these features will be tested and feedback will be given in Alpha and Beta testing.

Objectives:

The majority of the project time will be on developing the AI for the characters, employees and customers, and will be the key focus of the project. Some time, but a limited amount, will be used to develop the backbone/outline of the game to demonstrate the AI in a realistic environment and so that users can get a good sense of the final program once it has been fully developed beyond this project’s deadline.

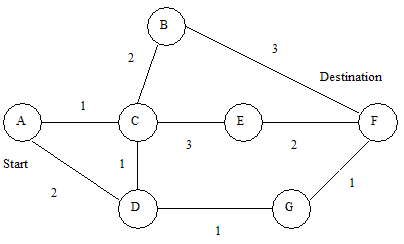
 There shall be investigations and research on different pathfinding techniques to find out which is best for the situation and to create the most realistic looking simulation. The A\* algorithm would be a good place to start but research will also be conducted on others such as Dijkstra’s.

Figure 2: Dijkstra’s pathfinding algorithm.

The backbone/outline of the program will be a top-down/bird’s-eye-view tile-based world which can be interacted with, and the player will be able to click on tiles/furniture/characters so that they can receive a realistic simulation of what the full game experience shall be like.

The player will not have a physical character to represent them in-game, but they will be able to interact with the world. Due to it strictly being a simulation, a pre-made scenario will be used as the demonstration, and will contain two employees, which will have randomly assigned traits, and the shop will be set up so the layout is the same each time. After that the spawning of the characters, their traits, and their shopping lists will all be randomised from a pre-built selection. This is so that if the user runs the simulation more than once, there is a very high chance that the way the simulation plays will be very different.

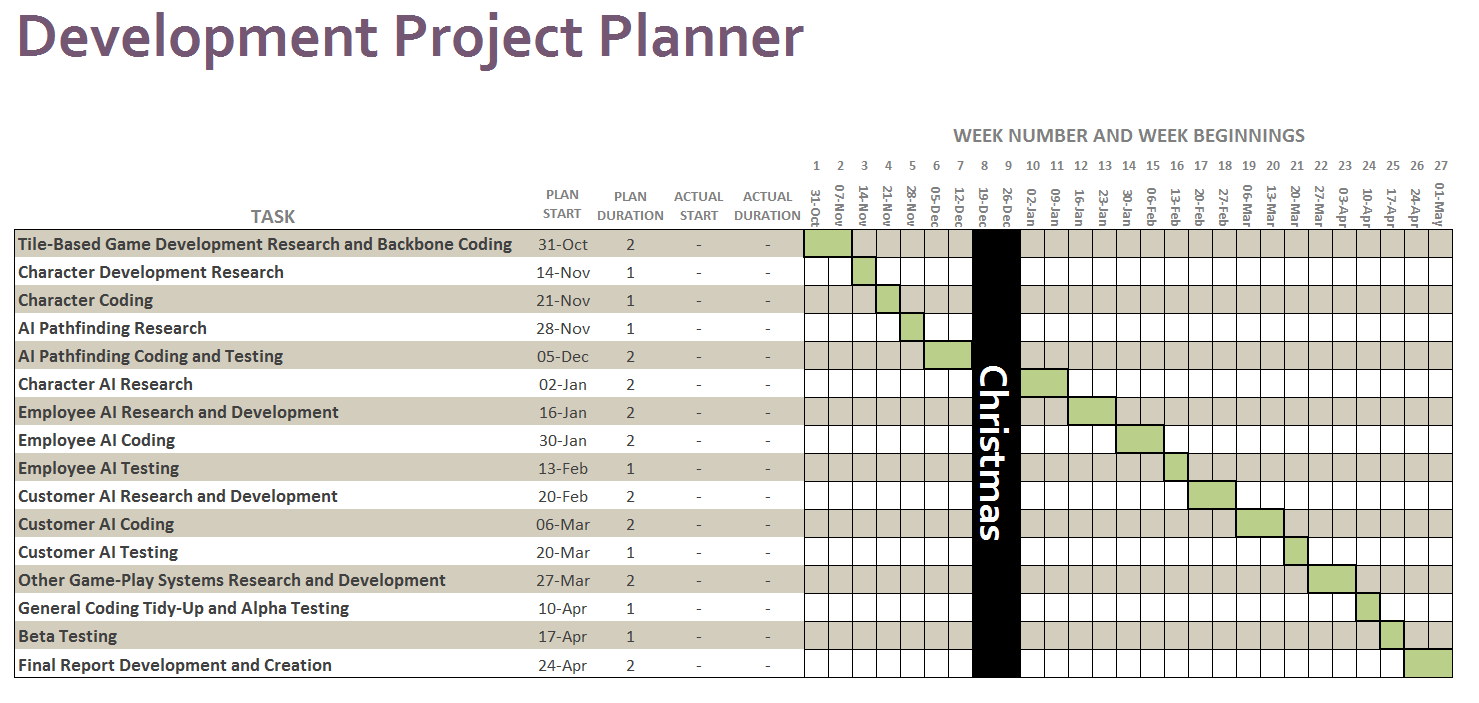
Methodology

The main stages will be carried out using the waterfall method. The base/outline of the program will need to be the first thing that is developed. Investigations will be carried out to implement world/tile interactivity and then development of the tile features such as furniture will be conducted.

The second part will be the character development, and will be where the characters are made and given traits. This whole step is required second due to the world needing to exist first, but the character’s existence is needed before any AI can be developed.

The third stage is the largest and most important part. The AI will be the core of the project and will require the most amount of time and research. Within the AI section, 3 sub-sections are required: Pathfinding, Jobs, and customer decisions. The pathfinding will need to be done at the start of the AI system, due to this being essential for every character regardless of whether they are an employee or a customer. The job AI and the customer AI are interchangeable as they are independent from each other. For this reason, the decision has been made to develop the employee AI second and the customer AI third. The employee AI is more important than the customer AI, and more complex, this is because the employee AI better demonstrates the final version of the game’s AI that the customer AI.

Research will be done throughout the development through finding documents about already developed management games, and tile-base games, and using their systems and information as a basis for the project’s development. General game-creation documents and reports will also be used when developing the pathfinding, and the other AI systems. Examples of good topics to focus on are fuzzy logic, finite-state-machines, and decision trees.

 The final stage of the project will be testing. There will of course be brief testing for each section throughout the project and on completion of each stage, which will all be documented appropriately, but also a final alpha testing stage at the end of the program development based upon the proposal, and other initial planning stages created throughout the project. After this, beta testing will be conducted where other people outside the project will be asked to play the game and give feedback on their thought and opinions. The game will be sent off to selected persons who are believed to be experienced in real-life shop environments, ideally in position of management. Upon receiving the feedback and testing results, a final report will be created based upon them and comparisons to the initial predictions and alpha testing will be conducted.

Here is the initial project gantt chart. These deadlines will be kept to as much as possible. All work will be documented in the project journal and so the dates can be compared.

Chapter 2: Background theory and Literature Review

Due to this program being a tile-based game, research will need to be done on general tile-based game development, as well as research on successful ways to create a backbone structure which will provide a secure and strong foundation for the rest of the more complex systems’ developments.

A peer reviewed literature paper Spuy (2010) [4] has been found and will be looked at as an example of a way to begin the development. It explains general tile creation, and how to use those tiles efficiently such as by creating an array of tiles to represent the world, and collision detection should be done on a tile by tile basis rather than object to object. Through this research, ideas have arisen about using tile sheets for the sprites used within the program, this would save memory and processing power due to one large sprite needed to be accessed instead of smaller sprites.

C# unity3D Code was discovered (Project Porcupine, 2015) [5], which links of a set of tutorials for a tile-based base building game. After looking through the code, the baseline code is very useful, and its logic is very similar to what is required for this project.

The project’s general set-up is that the visual aspects of the game are separate from the hidden game logic. It uses controllers to link the two together. These controllers are the only classes derived from Monobehaviour and so are the only ones that can use Unity’s GameObject functions and methods.

A class called WorldController is the center point of the code, and all the other classes, and instances can be reached via this class, either directly or through other classes, and it is the only class in the project that has been developed using a singleton system.

Other controller such as the FurnitureSpriteController and the CharacterSpriteController are used to link the furniture and character logic with Unity’s GameObject logic, respectively. These class collect data from standard C# class and use that data to place and move GameObject in Unity to where they need to be. There are other controllers their logic will not be used in this project so will be skipped.

The rest of the classes are not derived from monobehaviour, and therefore they represent the hidden game logic. The main class is named World and contains all the general game data such as the world’s tiles, lists that contain references to all the characters and furniture in the world, and the jobQueue.

Some of the classes are models and represent templates for different things in the game such as characters, furniture, tiles etc.

There is a set of classes that are used for the program’s pathfinding system, it uses the A\* pathfinding algorithm. This algorithm is generally known as the best algorithm as proven in a DePaul University technical report (Krishnaswamy, 2009) [6]. Research will be done at a later point to compare A\* with other algorithms and a decision will be made on the best algorithm for this project. If A\* is used for the development, then the A\* logic shown will be used as it already is integrated with the rest of the logic.

The Project Porcupine code is licensed as shown in Appendix 1.

A major influence on this project is the tile-based colony management game RimWorld (2013) [2], because of this, here is an insight into its systems and similarities with this project.

RimWorld allows the player to indirectly control characters in-game known as colonists. The player assigns jobs and the colonists fulfil those jobs based upon their own personal job priority lists, also set by the player. The colonists have moods, relationships, and thoughts. They also have traits which affect their speed, moods, relationships, and thoughts. The player can place a range of furniture, which must be built by the colonists using inventory. Due to the player not controlling the colonists directly, RimWorld has a very complex and integrated AI system which is used to make the characters seem life-like by changing their moods and thoughts.

The similarities with RimWorld and this project are as follow: uncontrollable characters, traits, relationships, and conversations. In RimWorld these systems are much more advanced and developed than what will be created in this project, however, the general idea and backbone of them will be developed throughout this project.

Chapter 3: Design, Development and Testing Process

Initial Coding

Using the Project Porcupine code as influence, the program began development with a WorldController, a World class, and a Tile class. The WorldController and World class are the center points of the rest of the code. After this the MouseController was developed, along with the Furniture class, FurnitureSpriteController, and the Furniture Actions class. The furniture actions class represents the Update function for furniture, however, due to not all furniture needing an update function, this class allows specific types of updates and only when the furniture needs it, such as when a door opens, or a piece of furniture moves.

All this coding was backbone, basic coding and put into place the basic functions needed to create these in-game objects and can be developed further later when required.

Research was done on general character coding, however not many useful literatures was found. Research was then done on RimWorld’s [2] character systems, such as their traits, moods and thoughts. Using this system, and the Project Porcupine character code, this project’s character code was started, and the Character class and the CharacterSpriteController class was created. These were again developed in such a way that the basics were completed, with lots of room and help for future development in several ways, such as pathfinding which was the next step in development.

Pathfinding

Many pathfinding literatures were reviewed and a few of them were focused on. By briefly explaining the major literatures found, reasons for the pathfinding choice made become apparent.

*Hybrid Pathfinding in StarCraft (2015)*

Hybrid Pathfinding in StarCraft (2015) [7] describes the use of A\* algorithms for long-range pathfinding, but also the use of potential fields for short-range and combat based pathfinding. It evaluates the possibility of replacing the potential fields part of the hybrid pathfinding with a system based upon flocking algorithms. The work of Olssen (P. M. Olsson, 2008) [8] addresses the issue of changes in the pathfinding graph due to the construction and deconstruction of buildings. Koenig and Likachev (S. Koenig, 2004) [9] (S. Koenig and M. Likachev, 2006) [10] have made contributions to the field with their work on real-time A\* pathfinding. Both works may be useful for this project due to some of the furniture can be moved, and so the characters will need to adjust their paths appropriately.

*Direction Based Heuristic Pathfinding in Video Games (2015)*

Direction Based Heuristic Pathfinding in Video Games (2015) [11] explains a general pathfinding concept and then talks about the two primary problems which are to find a path between two nodes in a graph, and to find the optimal shortest path [12].

It then explains A\* pathfinding. A\* is a generic search algorithm that can be used to find solutions to many problems, pathfinding is just one of them. A\* is the most popular and widely used AI pathfinding algorithm proposed by Hart, Nilsson and Raphael in 1967. Due to its simplicity, A\* is almost always the search method of choice. This is because A\* is guaranteed to find the shortest path on a graph.

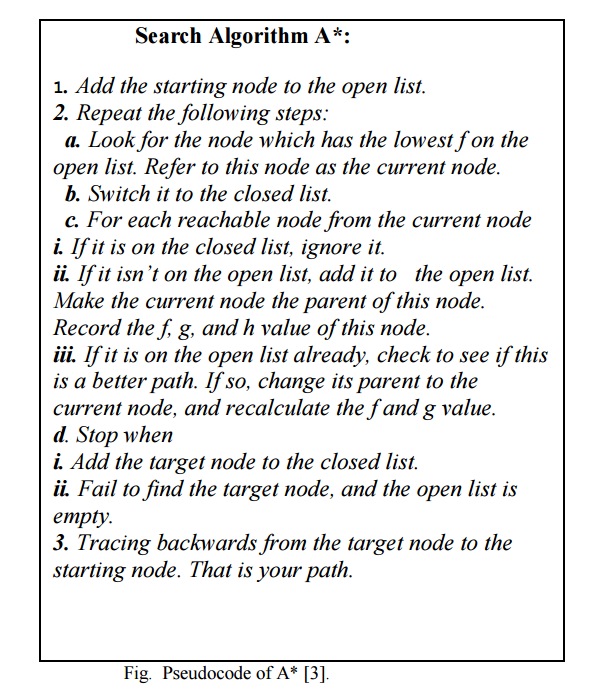
A problem with A\* is that a shortest path on a graph is not equivalent to the shortest path in the continuous environment. Another issue is when the map size if significantly large, it cannot find a minimum path to the goal state in a limited amount of time. For larger maps, A\* uses memory extensively. A\* uses a heuristic to improve on the behaviour to Dijkstra’s algorithm.

Figure 3: A\*’s Pseudo Code.

Next the publication goes on to talk about general heuristic features. The trade-off between speed and accuracy can be exploited to create a good balance. One way to construct an exact heuristic is to precompute the length of the shortest path between every pair of nodes. This is not feasible for most game maps. However, there any many ways to approximate this heuristic:

1. Fit a coarse grid on top of the fine grid. Precompute the shortest path between any pair of coarse grid locations.
2. Precompute the shortest path between any pair of waypoints. This is a generalization of the coarse grid approach.

Thinking about the links between this publication and the project, the downside of the large maps being slow does not matter in this situation due to the maps in the program only being 20-30 tiles high and wide. Also, only a maximum of perhaps 10 characters will be present in the word at any time, so the high memory demand is not a problem unless the world was expanded and more characters were present on the map, at which point A\* may become less viable.

*Uninformed Multigoal Pathfinding on Grid Maps (2014)*

Uninformed Multigoal Pathfinding on Grid Maps (2014) [13] is a publication focused on implementing multigoal logic into standard pathfinding algorithms.

There are two classifications of pathfinding algorithms; informed and uninformed. Informed involves the use of a heuristic function [14] to estimate the location of the goal. The direction of the pathfinding is guided towards the estimate, making informed searches typically faster than uninformed searches, but can be less optimal. Uninformed algorithms have been developed using different pathfinding models, such as iterative-deepening searches [15], boundary searches [16], bidirectional searches [17] and multigoal searches [18] [19].

The publication tested two algorithms: Dijkstra’s Shortest Path Multigoal Algorithm with Multigoal Boundary Iterative-Deepening Depth-First Search. The results showed that when multiply goals are required in the game, using the multigoal algorithm significantly decreased pathfinding times. There is an exponential increase in pathfinding times recorded by single-goal algorithms to search for multiple goals on open maps. This project does not require multigoal pathfinding due to the nature of the AI system that will be implemented, however, this is very interesting and will defiantly be considered if a more complex AI is developed with the ability to stack destinations for more efficient pathfinding.

*Pathfinding in Partially Explored Games Environments (2014)*

Pathfinding in Partially Explored Games Environments (2014) [20] talks about a very big problem with pathfinding and the fact that usually the characters in the game know everything about the layout of the map, and because of this, they can create perfect routes to their destination which is often unrealistic. It proposes a system using a hybrid approach [21] that allows characters to path-find as normal, but they do not know everything about the map. This system would require some way for the characters to ‘learn’ about the environment with, for example, line-of-sight. The character will detect changes in their environment as they move and adapt and change their pathing accordingly. For example, they could turn a corner assuming they can walk down it, and as they carried on walking they realised that it was a dead end and be forced to turn around and come back, which is much more realistic than them knowing the dead end was there without seeing it.

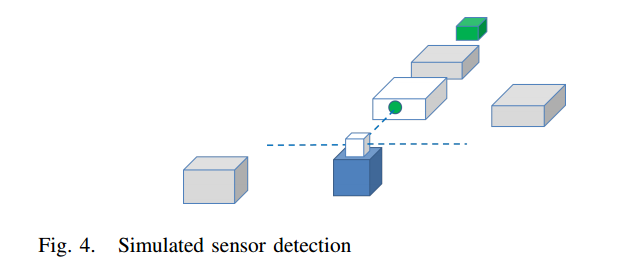
 The publication tests this system in Unity3D using ray casting as a sensor to ‘see’ the environment. The raycast works in a similar way to a sonar or LIDAR based system in that a single point is projected outwards. If the raycast intersects with an object, the system records the collision point. The character will be able to see any object the raycast intersects with.

Figure 4: Raycast ‘sight’ example.

This publication is very interesting and very relevant for this project. Having our characters not be able to see the entire map and require them to learn as they move would increase the realism and allow a move in-depth and advanced AI. The problem with this however, is the requirement for line-of-sight, as well as each character having their own perception of the world. Both these things require more coding, and additional systems be put in place for them to work successfully. This partially explored environments system was not implemented due to time constraints but the idea behind it was a huge possibility and will be one of the first additions for future work.

After pathfinding research was completed, the A\* pathfinding algorithm was chosen as the best for this project. Thought was given to the partially explored environments system, and it was decided that if enough time was available near the end of the project, then that system would be implanted along with line-of-sight. Due to Project Porcupine [5] already having a successful A\* pathfinding system, it was looked at and its basic outline was used. The pathfinding system was tweaked later in the project when moveable furniture was fully implemented.

The system in this way: when a character requires a new destination, they will use the pathfinding to find the shortest path from their current tile to the destination tile. Each tile in the world has a weighting which changes when certain furniture is on it, this means that a greater weighting causes the pathfinding to try and go around that tile if a shorter path is found; for example, a weighting of 2 for a tile means that walking through this tile is the same as walking through two tiles.

The final path is stored and when a character reaches the next tile in the list, it becomes their current tile and the next tile is used for the next movement. Tiles contain one of three ‘enterability’ states: yes, no, soon. Yes means that the character will walk onto the tile, no means that the tile is now occupied and the character will re-evaluate their path, soon means that the currently the tile cannot be entered but if they character waits then it should be able to be moved soon, such as a tile with a door. The character will wait, inform the tile they are waiting, and then furniture or character on that tile will perform their own logic for when a character wants to enter their tile.

Once this code was developed it needed to be tested. A basic character was created and a ‘GoTo’ button was created to allow the user to click on a tile, which then set the character’s destination tile to the clicked tile. Simple walking with no obstacles was tested, as well as testing with a single line of walls, and finally a closed off room with only a door to leave by. The character correctly stood next to the door until the door opened using its furniture actions, and when it was open the tile had an ‘enterability’ of yes instead of soon, and the character carried on their path. When the character left the tile, the ‘enterability’ changed back to soon and the door closed.

AI Development

The next section of the research and development was the most intensive and important part. In a similar way to the pathfinding, several publications were looked at, but only a few were focused on.

*Emotion-Based Synthetic Characters in Games (2008)*

Emotion-Based Synthetic Characters in Games (2008) [22] talks about emotions for characters in video games, and proposes a model using fuzzy logic to create facial expressions and body positions to represent emotions, and other characters used those emotions to learn about the character. It used basic emotions such as happiness, anger, fear, sadness etc. It talks also about keeping these moods around for a given amount of time, and so these moods can stack, or oppose each other if they happen at the same time.

The use of facial expressions and body positions is not relevant for this project; however, the idea of fuzzy logic is something that needs to be thought about and is used slightly in the moods and conversation responses in the characters in the program. The idea of keeping moods around was not relevant in this project as the characters don’t exactly have moods, they instead change their relationships with each other. The idea of them having moods, and the mood changes sticking around for a time is interesting and will be considered for future work. Having a bad conversation with someone could cause a character to be angry and may cause they to be more irritable to other characters, which is realistic.

*Towards the Design of Human-Like FPS NPC using Pheromone Maps (2013)*

This publication [23] begins by talking about different ways to model decisions for NPCs such as state machines, fuzzy logic, behaviour trees etc. and talks about their advantages and disadvantages [24]. The paper goes on to talk about different ways that characters can choose where to go or what to do. The example given was Quake 3. It talked about how if a certain location has had a lot of fire, and was being attacked more than other locations, then that logic should be worked into the AI to allow more solders to go to that location to defend, as they should know it is more likely to get overrun.

This logic can easily be used in a shop environment, but was not implemented into this project due to it not needing to be due to the very short space of time the simulation would run for. However, the idea behind it would be if it should be a normal day, with a normal number of customers, but today for some reason it was much busier due to some kind of event, then workers should use that information and perhaps not go off and do their normal jobs and instead stay near the tills or around the shop front to help the increased number of customers.

*Adaptive Behaviour Control Model of Non Player Character (2013)*

This paper [25] talks about a model using fuzzy logic to help define the behaviour in a non-player character. It takes the basic fuzzy logic system and creates an adaptive version. This allows the character to learn in an independent way from the other characters, meaning that the same event will cause different character to react differently even if other things such as traits are the same as the fuzzy logic values and logic changes the more the character interact with the world.

This logic is not useful in this project due to the limited time the simulation is run but is something very advanced and will create a very interesting AI for future work on this project.

*Component-Based Hierarchical State Machine – A Reusable and Flexible Game AI Technology (2011)*

This publication [26] begins by talking about general finite state machines (FSMs) and talks about how these are the most common in video games [27] [28]. These are used to model the behaviour of computer-controlled game objects to make the NPCs react to game events in the most natural and intelligent way possible. FSMs consist of a set of states which represent actions or behaviours and a collection of transitions from state to state.

It then proposes a new technique called Component-Based Hierarchical State Machine (CSHSM) which introduces software component techniques to the implementation of hierarchical state machines. It overcomes the limitations of Object Oriented Hierarchical State Machines (OOHSM) and has three significant advantages:

* Compile Time Composability. At compile time, new high-level and complex states are created.
* Design Time Configurability. CBHSMs are no longer completely fixed at compile time by programmers, and so can be configured at design time according to the game’s high-level design.
* Run Time Flexibility. A run time, CBHSMs can be reconfigured as needed. This feature frees CBHSMs from fixed hierarchical structure and greatly improves their flexibility and adaptability to the changing game environment.

The paper goes on to talk about standard OOHSMs and their limitations such as the defective ‘white-box reuse’ [29], and talks about decoupling the state and the context, but that requires an Asynchronous Event-Driven System (AEDS). It explains the process of establishing the AEDS and the implementation of the CBHSM system.

This paper, although useful for considering an advanced FSM system, is not relevant for this project. The complexity is unnecessary due to the simple simulation design the of program and so will not be used for the development of the AI. For future development, where the scenarios are not predetermined, this may be something that should be implemented.

*Game Coding Complete (2013)*

This book contains a huge amount of content about all aspects of game creation. Looking in the AI section provided lots of insight into the best and most known techniques such as: Finite State Machines, Decision Trees and Fuzzy Logic. Each one is shown and explained in detail and allowed a good source of knowledge to make an informed and correct decision into which approach was best.

With this book’s large amount of information and the rest of the AI research, the chosen system is the use a finite state machine (FSM) for the employee job code and the customer code. The different jobs that the employee needs to do works well with the different states needed. The FSM has primary states and secondary states. The primary states represent the major jobs required, and the secondary states are smaller tasks that the major tasks are made up of, for examples moving position or emptying a stockcage.

The characters’ relationships, interactions and traits will use simple fuzzy logic at add randomness to the simulation so that it isn’t the same each time, which would allow the user to keep repeating the simulation with different results to get a good grasp of the realism.

Coding and Development

The AI development began with creating employees that inherit from the Character class. These will use certain conditions to decide on the job they need to do. The FSM first begin in the Job class, but this was quickly abandoned as the employee didn’t have complete control over the machine, and the FSM now is implemented into the employee class. This means the machine acts as the employee’s ‘brain’ which is realistic. All characters have a DoThink() function but the logic will be overridden for the employees and the customers since they are in the shop for different reasons. The employee does a DoJob() function which is different from the DoThink() function as the DoJob() function only runs when the employee is not moving and needs to perform an action on the tile where they are.

After there were simply jobs, and the employees had simple logic, stock was added to the game. Stock can be added to furniture and characters can pick up stock. Character stock logic was then developed so that characters can try to pick up stock from furniture. They can try to pick up any stock from the furniture, a specific type of stock based upon the stock ID, or an exact piece of stock. Stock is a model class that acts as a template just like the furniture class or the character class.

Next was the advancement of the selection logic, which allows the user to select furniture and see what stock is on that furniture, this was done at this point due to the need for it when it came to testing the employee’s take stock code.

Once that was working successfully, then all the other furniture was implemented. The only moveable furniture is the Trolley, so that the employees can use the Trolley to take stock to and from the warehouse.

A very complex and important part of the development process was the moveable furniture. This caused the most amount of problems due to furniture needing to be moved by a character rather than independently, also the way the furniture is implemented it does not work well if it needs to be moved from tile to tile.

Characters can set a piece of furniture as their piece, and then move it if they require it to be moved. This was simple when just moving to another tile, but became complicated when a Trolley was in their way and needed to move it just to go around it. This links closely with AI since there are several ways the character could choose to move the trolley. Initially, the character would just push the Trolley until they reached their destination, but this quickly because unrealistic as it may make more sense for the character to move the trolley to the side and walk through the tile where the trolley used to be. This was when a small bit of pathfinding was implemented. The character would perform a floodfill from the trolley’s tile. This would return the nearest empty tile away from the trolley that is not in the character’s path to get to their task’s location. The character would then move to that tile with the trolley, once there they would return to the task they were performing.

After testing, this again became unrealistic as often the floodfill would return the tile behind the character, at which point the character would reverse and then once the moved back, once again the trolley would be in their way. This would repeat until the character literally trapped themselves in the corner of the store. The problem is that the character wasn’t thinking about where they were going to be once they did move. This sparked a complex function which used recursion to predict where the character and trolley would be if the character moved to the tile found by the floodfill. If the trolley would still be in the character’s way, the process would repeat, and a new tile would be found, once again reclusively checking where the character and trolley would be. This is all being done without the character moving anywhere. The recursion stops when either the trolley will reach a tile that will not be in the character’s path, or the character gets trapped. If the character won’t get trapped, then the character proceeds as normal, but if they would get trapped, then that tile they would have moved to gets flagged as an invalid tile to move to, and the process repeats until a valid tile is found. After testing this, it was found that it is a very realistic AI which successfully moves a blocking trolley somewhere logically and allows the character to pass.

Once the trolley movement logic was in place, the rest of the employee AI logic was completed. Buttons were developed to change the speed of the simulation so that already tested parts could be moved through quickly, and other parts could be tested without the wait. Once the employee code was close to complete, the customer code was added.

The customer FSM is much simpler and more straightforward than the employee one due to the simpler tasks the customers need to complete. The FSM still contains primary and secondary states, but they are not as plentiful, and did not take as much time to develop and plan. The customers spawn, and then are given a list of items they require, which they will then go and find. They begin from the start of the store and look in each shelf to find the item they need. Once they have found all their items, they floodfill from their current tile, until they find a tile marked as queue. All queue tiles have a number; the number is how far away that queue is from the checkout. If a customer is in the queue, and there is a queue tile adjacent to them, and the queue number is less than theirs, the check to see if it is free, and if it is they move there. Once at queue tile 1, they put their stock on the checkout, and the employee scans the stock, and the customer then picks the scanned items back up and pays and leaves when they have them all.

This is the point at which the line-of-sight/partially explored environment code would make the most sense. The customers may not have been to the store before and it would be realistic if they got confused and move around the store a lot, and sometimes missed shelves that they couldn’t see.

After the customer logic was completed, the employee logic was completed. It had to happen in this order as some employee logic, such as scanning and finishing a transaction, relied upon the customer AI being completed.

After this, general character AI code was added which caused the characters to not occupy the same tile, and this meant that when moving into a tile, in the same way a door causing the character to stop and wait, the character will stop and wait if the tile contains a character already. This would not always work however, as if two characters wanted to move to each other’s tiles, they would stop forever, and would never be able to move. This caused some logic to be added which meant that the characters would wait for five seconds, after which the character would try to find another way to their destination without going through the tile that is causing the problem. Sometimes there is no other way, and so they need to ask the other character to move so they can pass. This is the first character interaction required and so before this was done, the interaction logic needed to be developed.

Characters can request interactions with characters next to them, and could be rejected if the character receiving the request is busy, or doesn’t want to talk. This allowed the asking to move logic to be completed. At which point relationships and traits were added. Traits change the way that characters respond to other characters and there are positive, such as understanding, and negative traits, such as lazy. After this character selection was added to allow the user to select characters and see the stock they are carrying, and the traits they have. If a character is interacting then that can also be seen, along with the character they are interacting with, and the relationship level of that character.

The rest of the development was small additions such as a message pop-up. The scenario used in the testing was developed, with two employees, and the beginning and finish screens were created.

Testing

Once the scenario was created, and the program was ready, testing begin to find bugs in the game. The simulation was repeated numerous times and various bugs were found and dealt with. Unfortunate, a couple of significant bugs were found, and due to the short time remaining on the project, these could not be fixed fully. Instead, to make sure the simulation was ready for release to additional testers, the bug needed to be fixed using bad coding practises, which will be the first things looked at in the future to make sure the bad coding practises do not exist.

Once the simulation was run more than a dozen times, and no additional bugs were found, the alpha testing began. This was done by an in-house tester, James Jamieson, who ran the simulation and completed a similar set of questions which would be given to the beta testers. The results are as follows:

Chapter 4: Results of evaluation of the system or product.

Chapter 5: Discussion and Conclusions including critical reflection and future work.

References cited and listed using BU Harvard format.

[1] ChiVue Games, CG. 2016. Store Manage: Cellular Edition. V1.0. N/A. ChiVue Games. Available at: <http://store.steampowered.com/app/420910/Store_Manager_Cellular_Edition/> [Accessed September 2016]

[2] Ludeon Studios , LS. 2016. RimWorld. Alpha 17. N/A. Ludeon Studios. Available at: <https://rimworldgame.com/> [Accessed September 2016]

[3] Introversion Software, IS. 2015. Prison Architect. V2.1. N/A. Introversion Software. Available at: <https://www.introversion.co.uk/prisonarchitect/pc.html> [Accessed September 2016]

[4] Van Der Spuy, R. 2010. Tile-Based Game Design [online]. New York City, NY. Apress. Available at: <https://link.springer.com/chapter/10.1007/978-1-4302-2740-3_8> [Accessed 5th November 2016]

[5] Glaude, M. 2015. Project Porcupine. N/A. N/A. Available at: <https://github.com/TeamPorcupine/ProjectPorcupine/tree/5027adcc472a3489c21176924cbf8973f4cb2fc2> [Accessed August 2016]

[6] Krishnaswamy, N. 2009. Comparison of Efficiency in Pathfinding Algorithms in Game Development [online]. Chicago, IL. DePaul University. Report Number: 10. Available at: <http://via.library.depaul.edu/tr/10/> [Accessed November 2016]

[7] Hagelback, J. 2015. Hybrid Pathfinding in StarCraft [online]. 8 (4). 319 – 324 Available at: <https://ieeexplore.ieee.org/document/7063238/> [Accessed 28th November 2016]

[8] Olsson P. M. 2008. Practical Pathfinding in Dynamic Environments. Ai Game Programming Wisdom 4. Newton Center, MA, USA. Charles River Media.

[9] Koenig, S. 2004. A Comparison of Fast Search Real-Time Situated Agents. Proc. Autonom. Agents Multi-Agent Syst. (AAMAS).

[10] Koenig, S and Likhachev, M. 2006. Real-Time Adaptive A\*. Proc. Autonom. Agents Multi-Agent Syst. (AAMAS).

[11] Mathew, G. E. and Malathy, G. 2015. Direction Based Heuristic for Pathfinding in Video Games [online]. *In 2nd International Conference on Electronics and Communication Systems (ICECS)*, Coimbatore, India 26-27 February 2015. IEEE. Available at: <https://ieeexplore.ieee.org/document/7124867/> [Accessed 28th November 2016]

[12] Björnsson, Y, Bulitko V. and Sturtevant, N. 2009. Time-Bounded A\*. *In Twenty-first International Joint Conference On Artificial Intelligence (IJCAI-09).* 431 – 436.

[13] Lim, K. L. and Yeong, L. S. and Ch’ng, S. I. 2014. Uninformed Multigoal Pathfinding on Grid Maps. [online]. *In International Conference on Information Science, Electronics and Electrical Engineering*. Sapporo, Japan. 26-28 April 2014. Available at: <https://ieeexplore.ieee.org/document/6946181/> [Accessed 1st December 2016]

[14] Russell, S. J., Norvig, P., Candy, J. F., Malik, J. M. and Edwards, D. D. 1996. Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ, USA. Prentice-Hall, Inc.

[15] Korf, R. E. 1985. Depth-First Iterative-Deepening: An optimal admissible tree search. Artif. Intell., vol. 27, no. 1, pp. 97 – 109. Available at: <http://www.sciencedirect.com/science/article/pii/0004370285900840?via%3Dihub>

[16] Bjornsson, Y., Enzenberger, M., Holte R. C. and Schaeffer J. 2005. Fringe Search: Beating A at Pathfinding on Game Maps. Proceedings of IEEE Symposium on Computational Intelligence and Games, pp. 125 – 132.

[17] Moldenhauer, C., Felner, A., Sturtevant, N. R. and Schaeffer J. 2010. Singlefrontier bidirectional search. *In Proceedings of the Third Annual Symposium on Combinatorial Search*. Stone Mountain, Atlanta, Georgia, USA, 8-10 July 2010.

[18] Hongyun, L., Xiao, J. and Hehua J. 2013. Multi-goal path planning algorithm for mobile robots in grid space. *In The Proceedings of the 25th Chinese Control and Decision Conference (CCDC)* May 2013, pp. 2872 – 2876.

[19] Parodi, A. M. 1985. Multi-Goal Real-Time Global Path Planning for an Autonomous Land Vehicle Using a High-Speed Graph Search Processor. *In Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, vol 2. March 1985, pp. 161 – 167.

[20] Stamford, J., Khuman, A. S., Carter, J. and Ahmadi, S. 2014. Pathfinding in Partially Explored Games Environments: The Application of the A\* Algorithm with Occupy Grids in Unity3D [online]. *In 14th UK Workshop on Computational Intelligence (UKCI)*. Bradford 8-10 September 2014. IEEE. Available at: <https://ieeexplore.ieee.org/document/6930151/> [Accessed 3rd December 2016]

[21] Hartanto, R. 2011. A Hybrid Deliberative Layer for Robotic Agents: Fusing DL Reasoning with HTN Planning. Autonomous Robots 2011 Edition. Springer.

[22] Huang, X. and Peng, Y. 2008. Emotion-Based Synthetic Characters in Games [online]. *In Fourth International Conference on Natural Computation ICNC ’08* 18-20 October 2008. IEEE. Available at: <https://ieeexplore.ieee.org/document/4667798/> [Accessed 3rd January 2017]

[23] Yahyavi, A., Tremblay, J., Verbrugge, C. and Kemme, B. 2013. Towards the Design of a Human-Like FPS NPC using Pheromone Maps [online]. *In IEEE International Games Innovation Conference (IGIC)* 23-25 September 2013. IEEE. Available at: <https://ieeexplore.ieee.org/document/6659132/> [Accessed 3rd January 2017]

[24] Millington, I. and Funge, J. 2009. Artificial Intelligence for Games. 2nd ed. Kaufmann, M.

[25] Petrenko, T. and Tymchuk, O. 2013. Adaptive Behaviour Control Model of Non Player Character [online]. *In 15th International Conference on Computer Modelling and Simulation (UKSim)* 10-12 April 2013. IEEE. Available at: <https://ieeexplore.ieee.org/document/6527386/> [Accessed 8th January 2017]

[26] Hu, W., Zhang, Q. and Mao, Y. 2011. Component-Based Hierarchical State Machine – A Reusable and Flexible Game AI Technology. *In 6th IEEE Joint International Information Technology and Artificial Intelligence Conference (ITAIC)* 20-22 August 2011. IEEE. Available at: <https://ieeexplore.ieee.org/document/6030340/> [Accessed 8th January 2017]

[27] Thomas, D. and Hunt, A. 2002. State Machines. *IEEE Software*, 19 (6), pp. 10-12. November/December 2002.

[28] Wagner, F., Schmuki R., Wagner, T. and Wolstenholme, P. 2006. Modelling software with Finite State Machine: A Practical Approach. United States. Taylor Francis Group.

[29] Snyder, A. 1986. Encapsulation and Inheritance in Object-Oriented Languages. *In Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings* November 1986. Portland, OR. Pages 38-45. ACM Press.

Appendices

Appendix 1:

The Project Porcupine code is licensed under the GNU General Public License v3.0. This means that all the code can be used without hindrance assuming the copyrighted party be given credit.