Software Documentation
Quar-A-Team
6/7/2020

OVERVIEW

The project is a Pac-Man vs AI game built to run on multiple computer platforms. The user is first presented with a main menu where they can choose to play, see the rules, adjust volume, or quit.

The game requires two mazes: one for the player (user) and one for the computer (AI). The mazes are side by side. Each maze has a scoreboard over the maze with the score and the number of lives remaining. The game is timed using a single timer.

To play the game, the player controls their Pac-Man using the arrow keys. The AI Pac-Man is controlled utilizing a pathfinder to go to the nearest dot and avoid ghosts. The game is complete when either the Player or AI has collected all the dots, or one has lost three lives. Both the Player and AI have four ghosts that move around randomly trying to get the Pac-Man in their respective mazes. When a ghost captures Pac-Man, they (user or AI) lose a life. Upon losing a life, Pac-Man and the ghosts respawn to their starting positions.

When the game is complete, the user is shown a result screen with the final scores and time elapsed. The user can choose to play again or return to the main menu.

HOW TO USE SOFTWARE

1. Download Pac-Man zip file in the most recent release of the software on Git Hub
2. Extract the files from the zip file to a directory
3. Open the folder containing the extracted files
4. Run Pac-Man.exe

QUALITY ATTRIBUTES

Usability – A player should be able to understand how to play quickly in order to limit the time spent learning the game from trial and error.

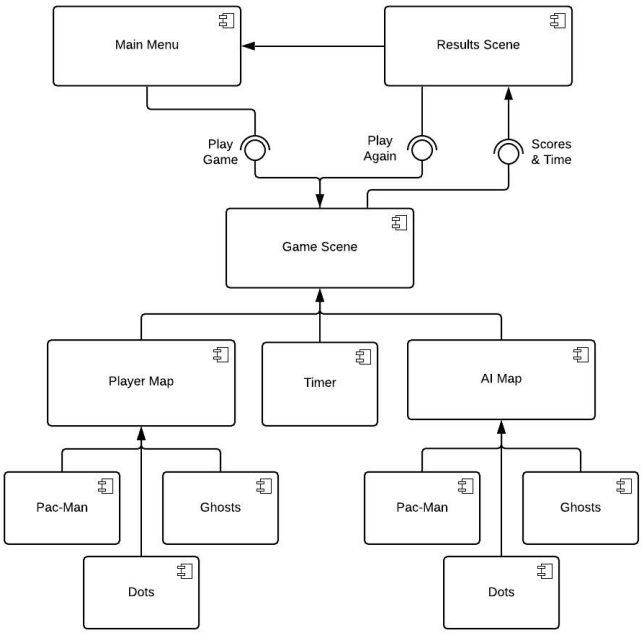
Reliability – The game needs to respond correctly to each keystroke the player inputs to have the game be fair for the player.

Integrity – The game needs to run smoothly so a player can appropriately respond to the game and to feel the game is fair.

SOFTWARE ARCHITECTURE

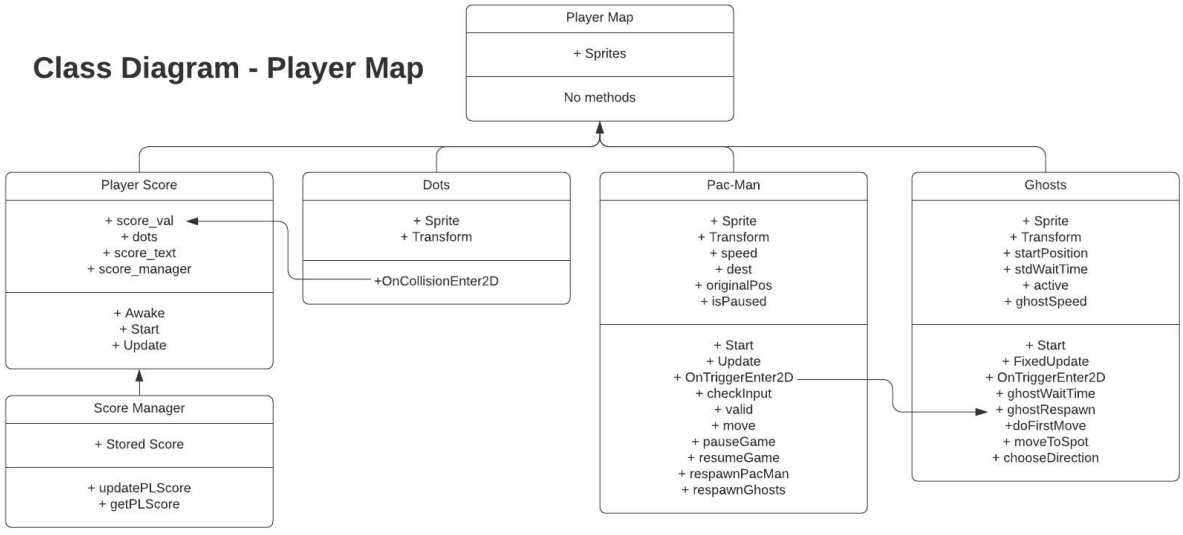
We utilized Unity for this project. Unity acts as a game engine and controls that GUI based on objects, scripts, and input from the user. In our project there are three main screens which are called scenes in Unity. The scene is the highest level in the software architecture (Main Menu, Game Scene, and Results). In the diagram below these are shown at the top of the diagram below.

Pac-Man Component Diagram



The user is first introduced to the Main Menu. The Main Menu scene contains a tutorial, ability to quit, and the choice to “Play Game”. The selection of play game leads the user to the Game Scene. The Game Scene is made up of both the Player Map and AI Map. These Maps are filled with Dots, Pac-Man, and Ghosts. All the necessary components of the game. When a game is completed, the user is brought to the result scene which gets the Scores and Time from the Game Scene. The user can go back to the Main Menu or select “Play Again” and go back to the Game scene to play.

To take a closer look at the details of how the player and AI works in a game scene, we will look at a class diagram for the player map and the components that make up the player portion of the game (see below).



The Player Map is made up of sprites. The maze is a static sprite that makes up the borders that the ghosts and Pac-Man must navigate. Pac-Man and Ghosts are sprites with their own script to define their behavior. Additionally, the dots fill up the maze and have their own script as well. Finally, the player score is a text sprite that changes as the score changes. The Dots, Pac-Man, and ghosts additionally have a transform which represents the location of the object at any given time.

All the objects act independently of each other except for when collisions occur. This is when two objects transforms are the same or are within a certain range of each other. For example, dots contain the script in "OnCollisionEnter2D" which detects a collision by Pac-Man. By detecting the collision, the dot is destroyed and results in the "score_val" for Player Score to be incremented and the Score Manager updates the score.

Another collision of importance is when Pac-Man collides with a ghost. This event runs the "OnTriggerEneter2D" function. This function performs multiple actions. It pauses the game, respawns Pac-Man, and it calls on the Ghosts' "ghostRespawn" function to respawn the ghosts.

Other than the functions detailed above, there is no overlapping of functions. The Player Score is managed with Player Score and Score manager and the score is displayed in the scoreboard section of the player map. The Dots remain in a fixed position and only perform the function detailed above upon collision. The Pac-Man contains scripts to move based on user input and performs the respawn function for both Pac-Man and the ghosts. The ghosts have scripts for random movement and additional functions to support respawn. This is identical to how the AI works, aside from the Pac-Man movement being dictated by AI scripts.

There are further granular details that have been left out to not add any more complexity. The various sprites have multiple 2D attributes for the Unity game engine to work appropriately and allow for the collisions to occur.

DESIGN PATTERN

The "Observer" design pattern is key to the proper functionality of the game. The game Dots are observing the gameplay for a collision between itself and Pac-Man. When the dot sees a collision occur, this triggers the score to increment either the player scores or the AI score, depending on the Pac-Man that caused the collision.

The player's Pac-Man and AI Pac-Man are also observing the gameplay for a collision with a ghost. The collision of Pac-Man and a ghost triggers the loss of a life, respawn of Pac-Man, and respawn of all the ghosts. In the instance of the player's Pac-Man colliding with ghost the game is paused as well.

Finally, the timer in the game is updated by the Score Manager based on the game time passing by. The time will not update when the game is paused.

FINAL STATE

Known bugs/issues

- AI Pac-Man will sometimes “teleport” positions when a ghost gets close
- AI Pac-Man will sometimes not recognize the last dot in the maze and stop
- Another….

Product Backlog

- Empty as we have completed the specifications for the project

FILE STRUCTURE

The project source code and objects are in the Assets directory separated into directories containing related items:

- AstarPathFinding Project – Scripts for AI movement
- Audio – Audio files
- Maps – Mazes and scoreboards
- Prefabs – Dots
- Scenes – Main Menu, Game, and Results screens
- Scripts – Source Code for all items except AI movement
- spritesAndGameObjects – Images of game objects (not including maps and scoreboard)
- TextMesh Pro – Resources for Text

The files in the Packages directory relate to the external libraries that were utilized in the project

HOW MEMBERS CONTRIBUTE

New members of our team or other developers can contribute by downloading Unity 2018.2.18f1 and cloning the Git Hub repository that Abraham is the keeper of. A new user would need to request to be a contributor. Any new contributor should know the following key items:

- Source Code is mainly located in the Scripts directory
- Source Code for the AI is in the AstarPathFinding Project directory
- If they want to change any of the UI, most of those items are in the scpritesAndGameObjects directory