Kelly Herstine and Karim Elzanaty

Jeff Timanus

AI Programming

September, 2018

1. Overall Architecture

The way that Karim and I structured this assignment was to try make a new player (pac) steering class as well as an enemy steering class. Each of these classes consist of their own direction and speed. On input, the GameApp class calls the PacSteering class to start its steering. PacSteering checks to see which button was pressed and then assigns the direction to the player to move in. It makes sure to check where there is a blocking value at the index that the player is currently at so that it doesn't clip through walls. This gives the player a more authentic movement. PacSteering uses A* in order to move itself. The enemy steering class has almost the same exact properties except it moves based on the state in the state machine that it is given.

We have four different states for the enemy AI and two different states for the Player. For the enemy AI, we have EnemyChaseState, EnemyFleeState, EnemyIdleState, and EnemyWanderState. For the player we have PlayerChaseState and PlayerWanderState. For EnemyChaseState, we give it the grid for it to use in order to move. From there, we get the center of the enemy position and where the player's current position is. We then use EnemySteer to move the enemy along the path towards

the player. We also do a check in this function to see whether the enemy is close enough to do damage to the player, if it is, we subtract 1 health from it every couple of frames. We also check to make sure whether the player has eaten the almighty candy and if it has, we transition to the enemy flee state so that it can move away. EnemyFleeState uses another version of A* that flips the heuristic, and only pathfinds 5 nodes ahead, rather than towards any real destination. We also check to see if the enemy is too far away from the player to actually chase it. We then set the transition to the wander state if this is the case. Lastly, we check to see if the enemy is at spawn, and if it is, we set its transition to the enemy idle transition. These checks are also used in the enemy wander state. The only difference between the EnemyWanderState and the EnemyChaseState is the the enemy wander state does not pathfind towards the player while chase does. Instead, in wander, we give the enemy a random direction at every intersection and tell it to go that way. EnemyFleeState is used when the enemy checks whether the player has eaten an almighty candy, and if it has, it starts the flee transition, it then changes back to wander after 10 seconds. Lastly, EnemyIdleState is used when the enemy first spawns into position. This is used for a certain delay with the enemy movement when it spawns, again, for a more authentic feel.

The PlayerChaseState is used to chase after the ghosts which is achieved by getting their ID’s through the unit manager. It selects the closest enemy, and pathfinds to that, regardless of where it is. Lastly, the PlayerWanderState is used to wander around the level. This was created similarly to how the EnemyWanderState was

created. It decides to take random directions whenever it gets to an intersection value in the map.

The player, enemy, coins, candy, and powerup all share the unit class. They all have their own unit maps and whenever any object needs to be deleted, it is sent to a “to be deleted” list so that once the update is done, it will delete the unit.

GameApp is in charge of creating and using all of the sprites that the game gives it. It is also where all of the data driven information is stored in the game.

2. Challenges faced in development

One of the biggest challenges that we face was the player movement. Kelly had originally tried to make the movement not based around pathfinding or grid movement. We ended up needing to scrap this. It took us both around three days to try and figure out player movement which was a major setback for both of us. Once we got this working, it was mostly easy. We had some issues with understanding how state machines worked but once we understood it, it was quite easy.

3. Areas where further improvements could be made

We could definitely make some improvements on collision in the game. It is a bit off at times and sometimes causes weird things to happen with the enemy state machine. There is also a single problem where the player AI will pathfind into the enemy spawn and become stuck until manual control is resumed.

4. What did each of us work on?

Kelly: I helped work/solve the player movement, I did the state machine transitions as well as the powerups, almighty candy, enemy respawns, and coins. I also helped Karim a bit with the state machines for the enemy AI.

Karim: Helped get a lot of the underlying movement structure to work. Did AI states for player and worked with Kelly with enemy, but not the transitions.