Decisions made

In our game, we made several decisions about how best to implement it. First, we had to decide on the mechanics of the game. We wanted it to be a top down RPG game with two levels: the first floor and the second floor. Each floor had to be set up with multiple rooms with enemies for the user to defeat. Once the user defeated all of the enemies, they would pick up a key and unlock the boss. To win the game the user must also defeat the boss. The user loses if they lose all of their health before defeating the boss.

For the health part, each enemy has their own health levels and the player does as well. When the player enters a room, the enemy starts to launch projectiles and the player has to evade those projectiles while launching projectiles of their own at the enemy. This will continue until all enemies have been defeated. Once this has happened, a key will appear which will grant the player access to the boss room.

Challenges

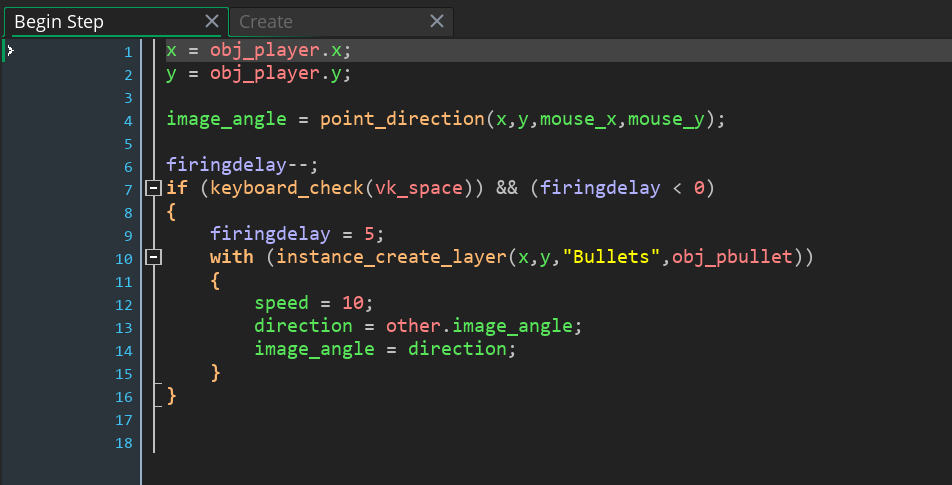
One of the challenges in this project is coordinating all of the different parts of the project. We did not realize at the beginning that so many of the elements of the project were interconnected. For instance, the health of the player and the enemies were interconnected with the player and enemies themselves. This made it difficult to split up the work. Another challenge is that we sometimes had to edit the same file which made it tricky when uploading to GitHub. If two group members had to edit the same file GitGub would not allow that file to be merged. We had to then go through and let one person edit their version to include the other person’s edits.

Core algorithms/routines

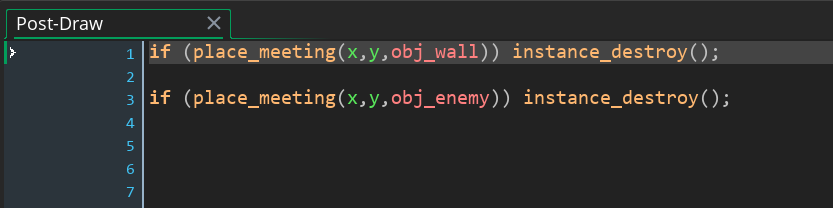
We did not have any core algorithms but we did have some core routines. As seen in Figures 1-3, the code does not have any specific algorithms. Game Maker structures everything so that it loops everything based on the frame rate of the game. Since the game is 30 frames per second(fps), it loops through everything 30 times in 1 second. This makes it so that the game can be continuously played. Each of the code snippets below has a list of functions that perform their functions based on the frame of the game.

For example, in Figure 1, the code shows that the player object launches projectiles in the direction of the mouse cursor. This is because we used the built in function point\_direction(). This allows us to define the direction that we launch the projectiles from the player. The same is for the enemy it is just that the enemy launches the projectiles at the player.

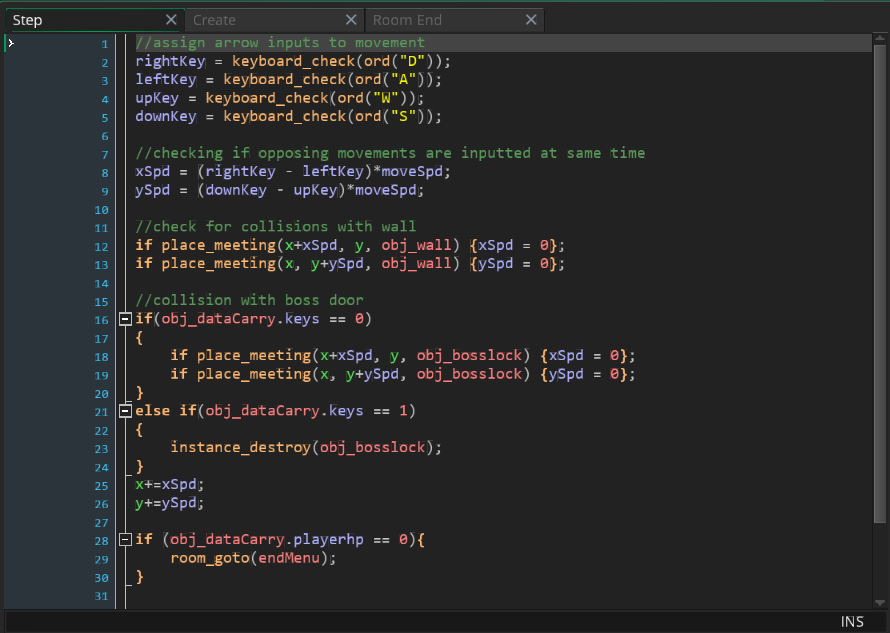
Annotated code snippets



*Figure 1: This picture shows the code for how the player launches projectiles. It uses a separate object called pgun and it sits at the same x and y coordinates as the player but it points in the direction of the mouse cursor. The projectiles are launched on a separate layer called “Bullets”. There is a version for the enemy. The difference is the enemy shoots in the direction of the player.*

**

*Figure 2: This picture shows how the projectile destroys itself when it collides with the wall or an enemy(for the player projectiles). This is similar for the enemy projectiles. The only difference between the player and enemy projectiles is that the enemy projectiles destroy themselves when they hit the player.*

**

*Figure 3: This picture shows how the player moves. Lines 2-5 check to see if the “A”, “W”, “S”, “D” keys are pressed and the code then moves the player in that direction. It also tells if the player hits the wall and keeps it at the same spot.*