Death Blocks

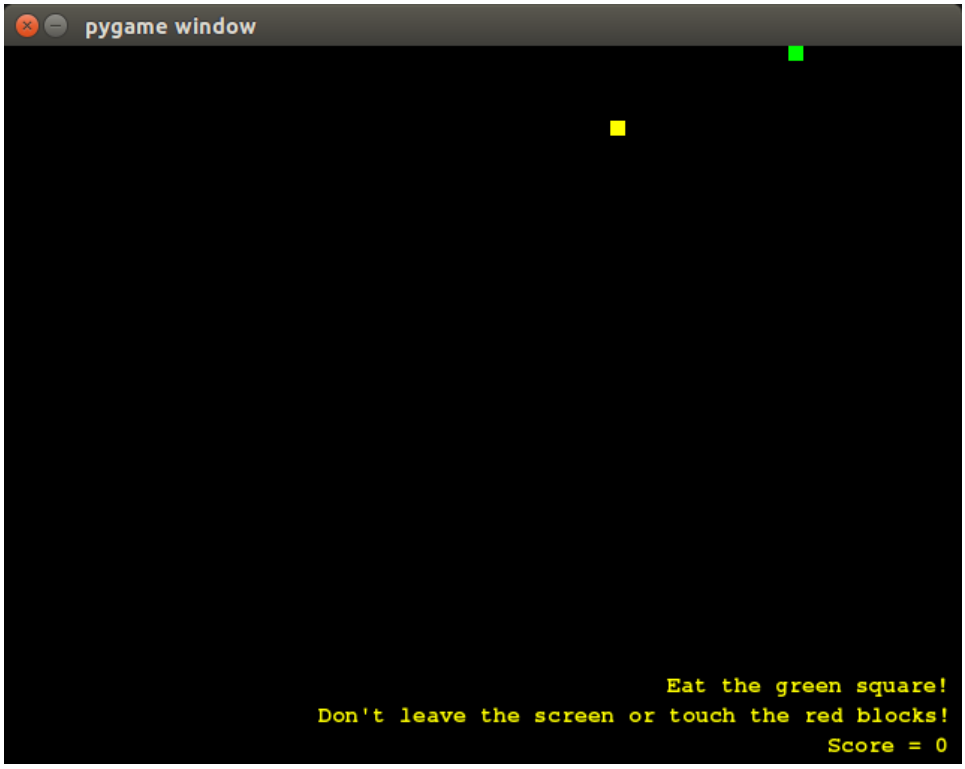
by Emily Mamula and Zarin Bhuiyan

Abstract

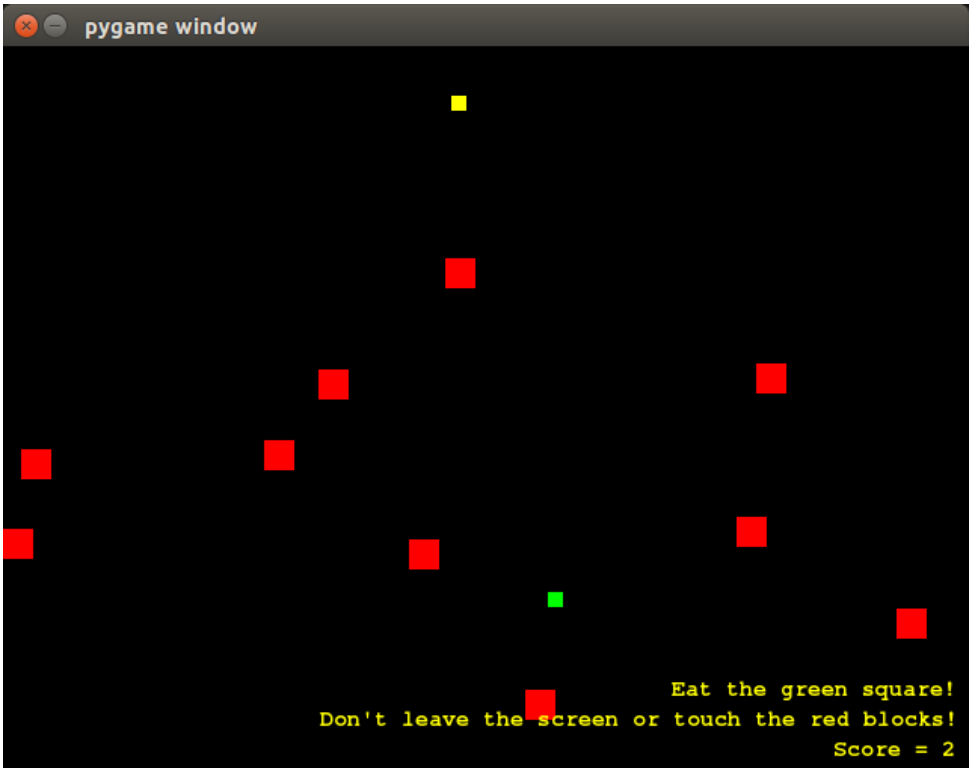
Our project, Death Blocks, is an interactive game that was created following some of the components that make up the classic game Snake. It relies on user input and program responses to function. Some of the computational skills we included are object-oriented and event-driven programming and simple physics. We used the Pygame library because it had many of the functions that we would need to implement this game. The rules of the game are simple: the yellow block must move across the screen and eat a green block to score a point. The yellow block cannot touch the edges of the screen or any of the red obstacle blocks that pop up or it dies and the game ends.

Results

We used the Pygame library to create an addictive and fun game that is a twist on the classic game ‘Snake’. Our game is interactive and relies on user input to run. The arrow keys are the controls that the user uses to move the main object, the yellow block, across the screen. The player’s goal is to get the yellow block to “eat” the green blocks that pop up, while avoiding touching the edges of the screen or the larger red obstacle blocks that also pop up. The speed of the yellow block increases with each point the player gets, which makes this game harder (and more fun). We also added an anxiety-inducing background theme song to add onto the intensity of the game. The score tracker keeps track of how many green food blocks have been eaten.



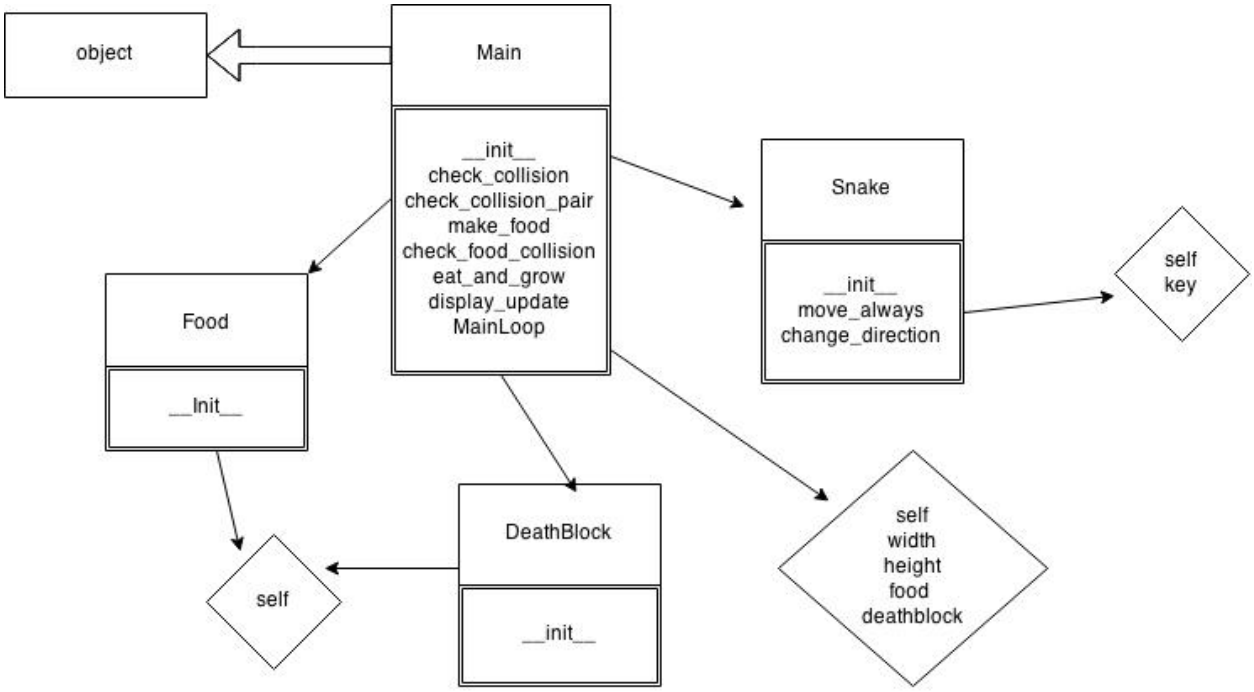
This is the start of the game. The bottom right displays our simple instructions and the score counter. The yellow block moves constantly and has to eat the green block.



For each green block eaten, five randomly placed red obstacle blocks appear.

Implementation

Death Blocks UML Diagram



The final implementation of our game consists of four classes: Player, Food, DeathBlock, and Main. The first three classes are all object classes that contain a variety of attributes and functions associated with those attributes. The last class is responsible for the game play mechanics. The objects are passed to the Main class and play a role in a number of functions within the class.

The data structures within the classes are mostly rectangle objects. Specifically for the death blocks, this includes a list of many rectangle objects as more death blocks are initiated. There are multiple instances of rectangles being created, referenced, and drawn. When the player rectangle is moved, it is done so at a certain speed that increases with the score in a way that corresponds with the last user-inputted direction. There are also many different collision checks. These include checks for player/food, food/death blocks, player/edge, and player/death block collisions.

Of the many design and implementation decisions we had to make, one was how to ensure that food rectangles could not generate within death block rectangles. We could either have used a while loop structure that may have been more simple but would be harder to follow immediately or a system of short called functions, including a recursive collision check.

We chose to go with the many shorter functions because it not only helped to make our code more readable, but also gave us the opportunity to reiterate some of the previous project topics.

Reflection

From a process point of view, implementing our program bit by bit went well. We made sure to work on one thing at a time and have it functional before moving onto the next part. This also helped keep our code clean and relatively easy to read. We could have improved upon learning the Pygame library better by experimenting with different possible functions rather than going with what works. We could also have practiced more with object-oriented programming before taking on this project to familiarize ourselves with it.

Our project was definitely appropriately scoped. It was challenging in a way that forced us to learn to use object-oriented programming and learn the depths of Pygame. This will definitely come into use in the future when we need to construct programs that focus less on the logic of implementing and more on how things are being implemented. Classifying things into classes and subclasses helps with this method of thinking.

Our plan for unit testing consisted of continuously running the game and being the user. It helped us see what our next steps would be and what needed to be improved. Pair programming worked well for this project. We worked together on the code, instead of having each person do parts of it on their own. This both sped up our coding and allowed us to learn from each other. In the future, we would plan to spend even more time pair programming.