Sam Myers
Matthew Beaudouin-Lafon

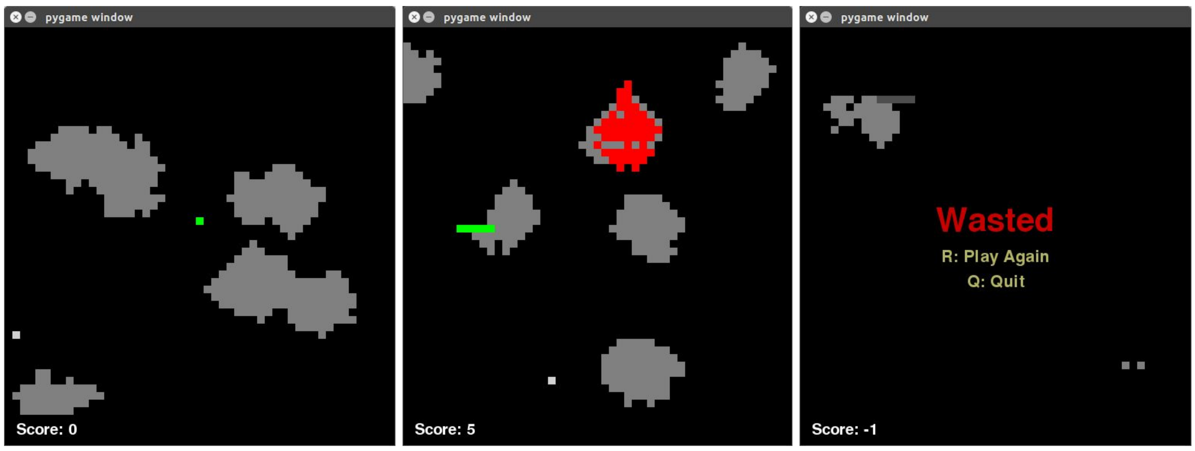
SNAK3D

Overview

SNAK3D is simply the classic game Snake with another dimension. The player always acts in two dimensions, but can rotate the view into an orthogonal plane at any point.

Results

Our game works (albeit slowly).
We made a 3D environment for the snake to move around in. Food generates when the game starts and every time it is eaten. Walls are procedurally generated in the beginning of the game. The player mainly moves in two dimensions (and sees two dimensions at all times), but can change the orientation of the map. That is to say, the player is always acting in a slice of the cube. The score is decremented every time the player moves, but every time he eats food he gains 10 points. The goal is to make the player think wisely about his moves to optimize his score.

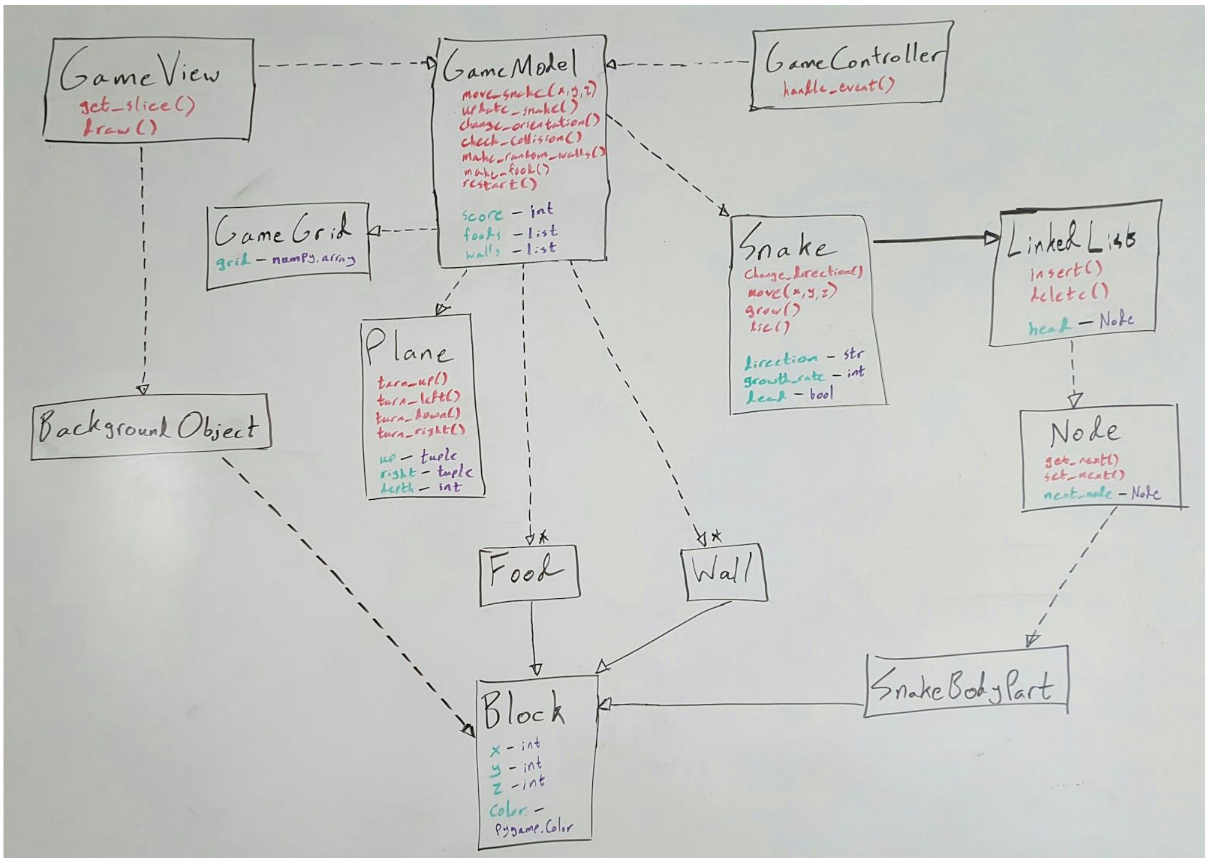


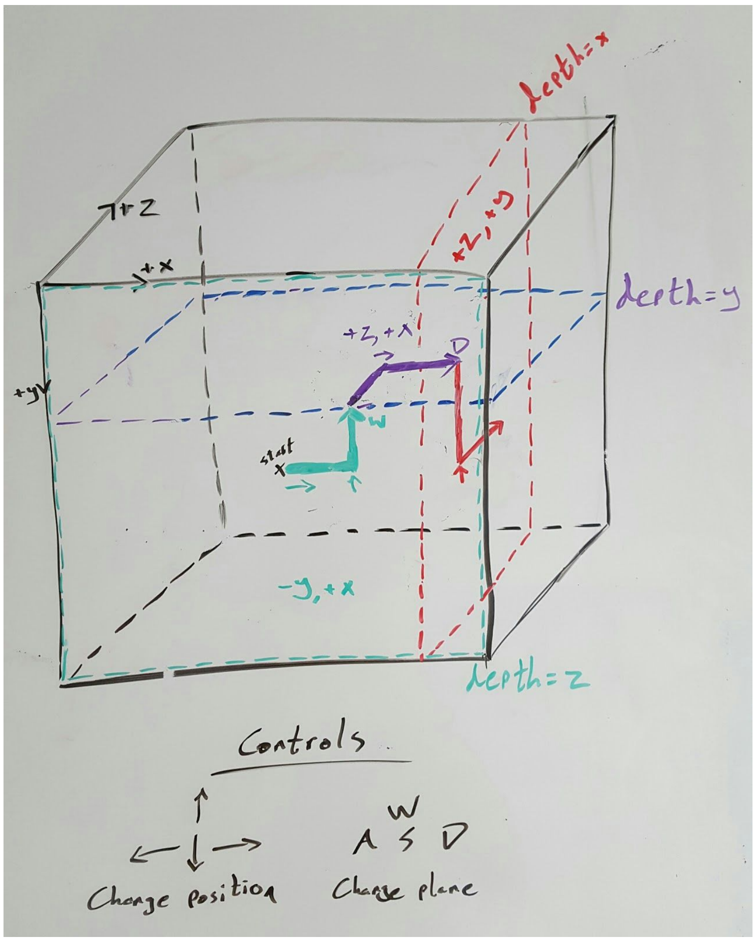
Implementation

The GameModel class stores everything contained in the current game state. This includes the snake, the wall and food blocks, the plane in which the snake is currently moving, and the player's score. The model also contains a "grid", which is simply a 3D array that stores all of the blocks in the game. The GameView class uses the data from the model to create a slice for the player to view (it does this by iterating through the grid), which is rendered to the screen in each frame. The orientation of this slice is determined by the information contained in the model about which direction vectors are "up" and "right" relative to the player's view. The view also projects objects outside of the current plane in grayscale by iterating through the other slices of the cube and placing a proxy "BackgroundObject" into the view grid in the appropriate location. The GameController class takes keyboard input and changes the model accordingly.

Moving the snake takes place in multiple stages: in each keyframe, first the controller changes the direction of the snake according to keyboard input, then the model moves the snake by one block in the appropriate direction (by calling the snake's internal move function), and finally the model updates the snake's position in the grid. The snake itself inherits from a linked list class, which consists of container "Nodes" that store individual snake parts. It moves by targeting a specific block, adding a new head to the linked list in that block, then deleting the tail from the list.

Changing the orientation of the plane in three dimensions uses some cross-product magic to generate new direction vectors from the previous one. Because its direction is relative and its movement is based on the direction vector, the head of the snake will always be moving within the current plane. Also, when the snake encounters an edge, it rotates 90 degrees onto an orthogonal face of the cube.





Reflection

From an object-oriented standpoint, we feel like our project was really well structured. The classes and functions are implemented in such a way that they intuitively interact with each other and adding components to the code is relatively seamless. We have location of elements (wall, snake and food) both in the grid and in the class, which looking back may be suboptimal in terms of OOP.
We underestimated the mathematical difficulty of implementing the game in 3D in such a way that the orientation of the screen makes sense. We possibly could have done more unit testing (we totally didn't write all of the 2D version without running any code before it was done).

When we made the 2D version, we pair programmed since we were scaffolding the project. After that, we delegated functions but continued to work in the same place at the same time. This helped us agree on implementation and conceptual approach. In practice, I don't think the work was perfectly evenly distributed simply due to the range in task difficulty. In future projects, it might be helpful to take some time to identify how long various tasks would take to more evenly distribute the workload.