**Introduction**

Title: Ant Simulation

Group Members: John Wiesner

**Overview:**

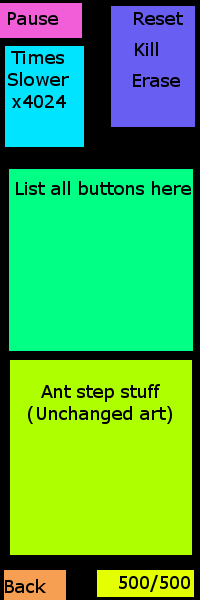
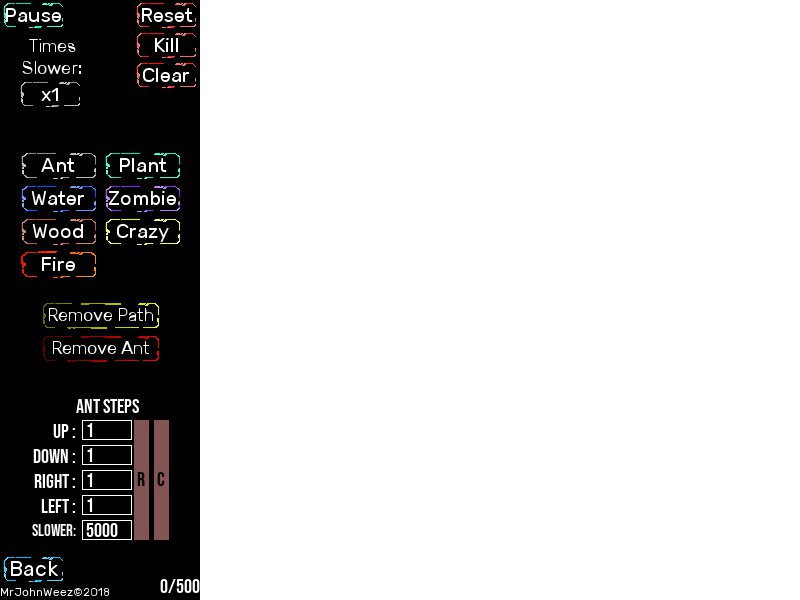
Many hours have gone into this project and in return a fully fleshed out game with a downloadable executable was produced. This project is based on the concept of Langton’s Ants: if an ant was the size of a pixel, the ant would turn and move left/right dependent on what color pixel it was on. This creates a pattern like path on a screen when several thousand steps are executed. Taking the interactive pixel elements from a game called The Powder Game and merging them with the Langton’s Ant concept, I developed the game/simulation that is presented in this documentation.

**Application design:**

The only 3rd party API used to create this application game was Pygame. My self-goal and challenge for building this game was to make the entire game from scratch: meaning using no other 3rd party code, graphics, nor Pygame examples (Coping a tutorial and changing a few things) were to be used. Setting these rules provided a more open-ended development and game design process that would otherwise be influenced by a Pygame tutorial or non-original game.

There are three main components of an enjoyable playable sandbox game: Simplicity, Creativity, and Learnability. Developing a program that satisfies these requirements would take many years to prefect. Knowing this I chose to take a step back and look at what other popular sandbox games have done and tried to adapt their concepts to my constraints and game idea.

Before placing buttons, text and boxes over the screen, I decided that a simple left sided menu would be the best for this game design (Similar to The Powder Game layout). All the buttons would be located on one side and there would be no need for searching the screen to find the correct button. This creates a simple UI.

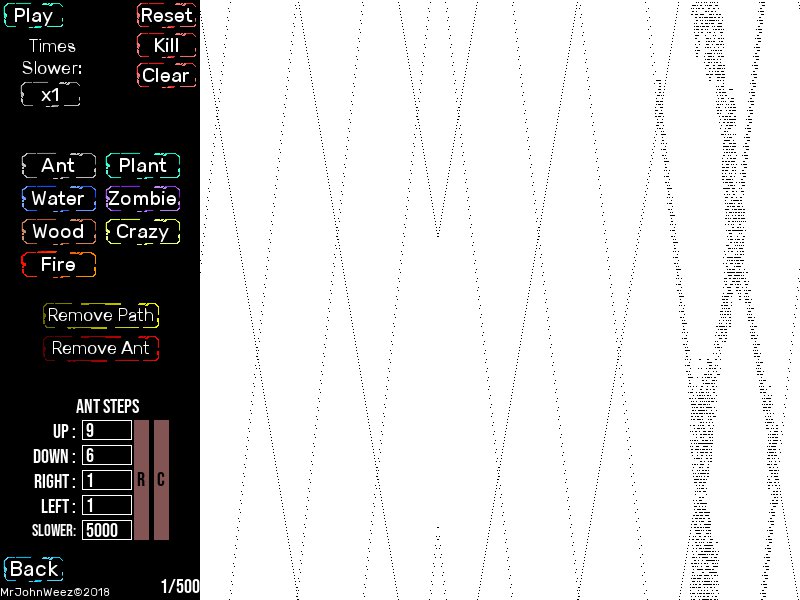


Concept layout vs final layout

Having this simple layout saved time when coding because there were no adjustments needed and code did not need to be rewritten. All buttons are organized into categories and have corresponding colors to make it simple for a user to distinguish functionality.

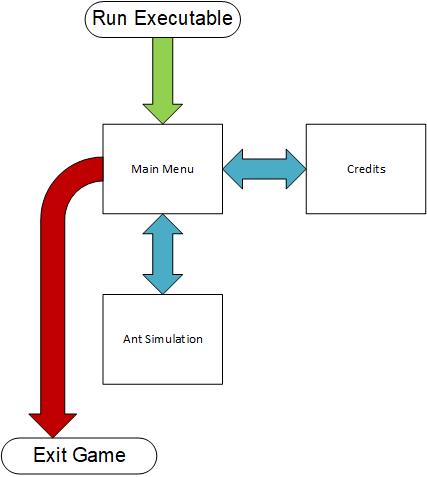
The learnability of this game applies to the way the ants interact with each other on screen. Each ant has its own properties and it is up to the user to discover what they are. There is no in-game description what to do nor how the ants work. This curiosity is what draws users’ attention and makes them want to keep playing. For the minds who want to know more about each ant’s properties there is a help button on the main menu that brings you to a website showing what each ant’s properties are.

This game provides its users with an opportunity to be creative. The user is in control of how many ants are on screen, which ants live, which ant paths should stay on the screen, how fast the simulation should run, and even how some of the ants can move. Using the provided existing ants a user can create their own ants by changing the ant step properties. Many unique patterns can be created by doing so creating designs like the one below:



Overall this Python application/game meets the three previously set requirements defined above in a respectful and elegant way. This hopefully provides all users with many hours of enjoyment and delight.

High-level diagram of the structure and flow of the application:



**What was accomplished:**

I knew very little going into this project: no Pygame experience nor how to use python classes properly. After spending many hours working on this game I have learned how to create a basic game using the Pygame API and how to best use classes within python. I broke all my code into modules because having 2,500 lines of code all in one file is not programing best practices. This also allows me to use the modules I created in other future projects increasing portability and scalability.

I made 8 different modules with a main module as the main script. This main script is called AntSimulation.py and \*contains the main menu, credits, and ant simulation game setup. Below is a description of each module (These can be found in code also):

**Ant.py**: Ant object types and all moving logic/ ant instances are created using this

**AntStepVar.py**: Ant.py uses this type of data structure for how to move on screen

**Colors.py**: Predefined color types and a function to convert a color to a dark/light shade dynamically

**CustomPath.py**: A function that grabs the OS path and combines it with a given asset path.

**ImageManager.py**: Allows for easy manipulation of Pygame images on screen

**Interactive.py**: Holds all types of buttons and text boxes. All interactive UI.

**Text.py**: Creates a text object that can be moved on screen and updated

**CustomMath.py**: A basic clamp function that should be in python’s default math library.

\*Note: There is one class located in the top of the AntSimulation.py file but is not a fully dependent class meaning it is dependent on variables within the files it lies in. It did not make sense to spend time on it because there is only one instance in the whole game. There are also many classes within the body of the same file, but these are specific classes and will most likely not be used in other games.

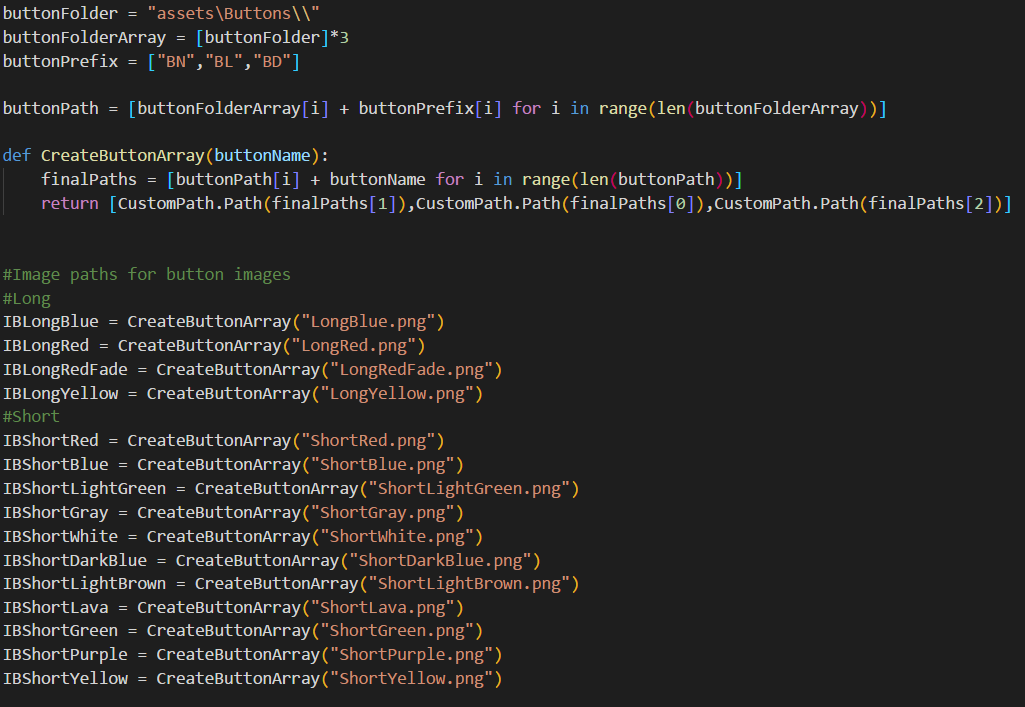
The custom classes allow for very easy instancing of objects. To create an interactive button two lines of code are needed. To create an ant there is only one line of code needed. Colors can be used as names instead of RGB tuples. Asset path names can be generated in a very short line of code instead of a block. Having all these objects makes moving objects on screen very easy and time saving. Especially when scaling the screen. Creating these classes took a long time and there were some tough times.

I would say the most time creating classes was spent in the Interactive.py script. This was because there is a lot of math needed for transforming all the Pygame Rects to display objects on screen. Also getting an interactive text box that allows the user to enter any number but also change that number by hovering over the box and use the scroll wheel, took a while.

Another class that took a while to create was the Ant.py script. All ants derive from a base class that has functions like: get/set pos, move types, and a basic update function. Since all the ants have different properties each ant update loop had to be written and changed to create these effects. The longest update loop is the Zombie ant because it must check which ant’s path it is on and correspondly turn into that type of ant a second later. (All ant properties are documented in the code under each ant class type).

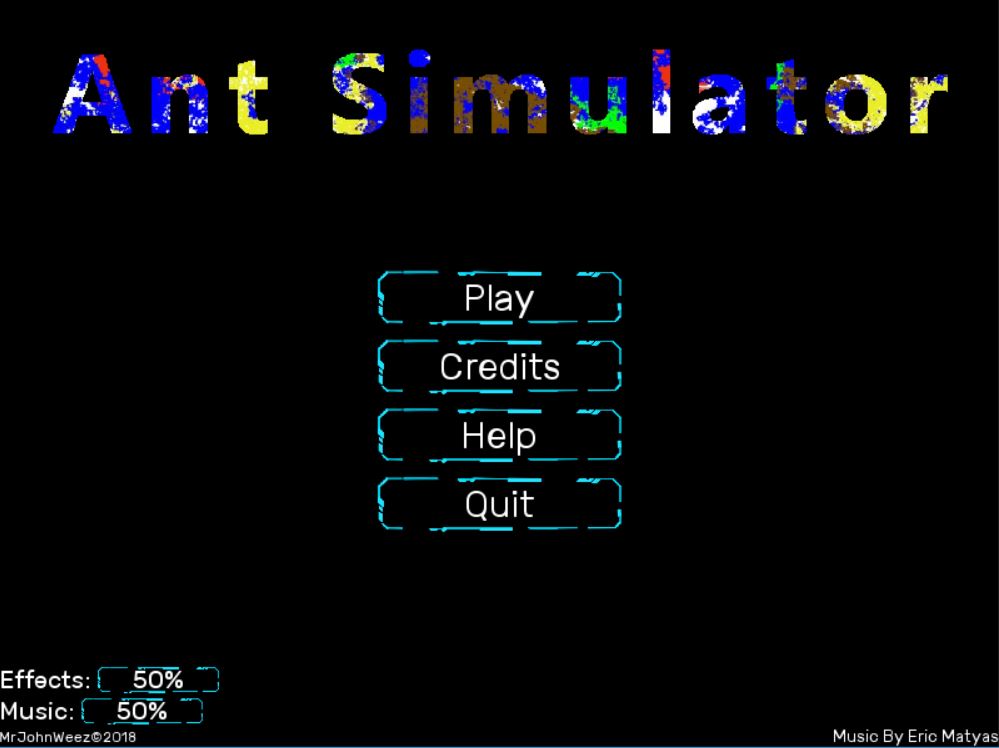
Another problem that I ran into, many times, was the game was slowing down to much. My simulation must run as fast as possible and the base line for my game is about 10,000 fps. This allows the ants to move every pixel and calculate each step in real time. Whenever a portion of the Pygame screen is updated it increases the computation time drastically slowing down the simulation. To prevent this, highly optimized code needed to be written so all buttons, texts, and ants are only updated when they need to show a different image on screen. The location and buttons of the mouse was very crucial to detecting if part of the screen needed updating. Completing all the optimization had about a 250% increase in game/simulation performance.

My most proud successes are compliments to the toughest parts of the project described above and the way I was able to solve and achieve the needed game logic. An interesting fact about this project is I made all art and sounds used in the game. Every button color was a separate image and every sound was created using my laptop and things around my desk. Note: I did not make the 3 tacks of music in the game though.



*Image above*:(How button image paths are dynamically generated in ImageManager.py)

One last success was the way I was able to let the user control the volume of the game sound and music. This took a while and every sound had to be created, exported, and loaded by hand into a list (Located at the top of AntSimulation.py). Below is a screenshot of the volume buttons:



Possible future improvements could be to use a single grayscale image of the buttons and colorize them dynamically. This would allow for easy changes in code to change the color of all buttons. This was not done originally because there was not enough time to figure out how to change image colors with code. More ants could be added in the future and way of showing the user what their current active ant selection would improve the user experience. The last improvement could be to remove all global variables from the AntSimulation.py file and pass them all in as references instead. This would make some of the classes and functions more portable in the future.

**Division of Work:**

Even though I worked alone I still divided the project into phases:

Phase 1:

Create a design on what the game should look like and what the game should feel like. This is thinking about what the user experience would be if certain features would be included.

Phase 2:

Create the necessary Pygame objects to have a functional game. This includes: mouse over buttons that enlarge when the user hovers over them, moveable images on screen that also scale properly to the window size, and clickable and editable text boxes that a user can edit.

Phase 3:

Slowly create all the game logic and simulation effects. The core of the game.

Phase 4:

Polish the game layout. Making sure the are no broken buttons and all features are displayed and are functional. This also ensures the game is able to work in full screen and scales properly.

Phase 5:

Add game art to make the presentation better and the user feel like the game is a true game and not just a proof of concept.

Phase 6:

Add sound a music

Phase 7:

Test the game like crazy to discover any bugs and fix them.

Phase 8:

Create executable and Game release

**References:**

**Topics that gave me inspiration:**

Langton's ant: <https://en.wikipedia.org/wiki/Langton%27s_ant>

Powder Game: <https://dan-ball.jp/en/javagame/dust2/>

**Libraries\Programs:**

Pygame: <https://www.pygame.org/news/>

Cx\_Freeze: <https://anthony-tuininga.github.io/cx_Freeze/>

**Tutorials/Code/Assets:**

Pygame Tutorials: <https://www.youtube.com/watch?v=K5F-aGDIYaM>

Interactive Textbox: <https://stackoverflow.com/questions/46390231/how-to-create-a-text-input-box-with-pygame>

Background music: <http://soundimage.org/>

**Inspirations for ant types:**

Fire ant:

<https://en.wikipedia.org/wiki/Fire_ant>

Zombie ant:

<https://en.wikipedia.org/wiki/Ophiocordyceps_unilateralis>

<https://www.wikihow.com/Walk-Like-a-Zombie>

Crazy (Acid) ant:

<https://en.wikipedia.org/wiki/Rasberry_crazy_ant>

Link to repository: <https://bitbucket.org/MrJohnWeez/ant_sandbox>

Download the game executable: <https://mrjohnweez.weebly.com/ant-simulation.html>