Interactive Programming – Writeup & Reflection

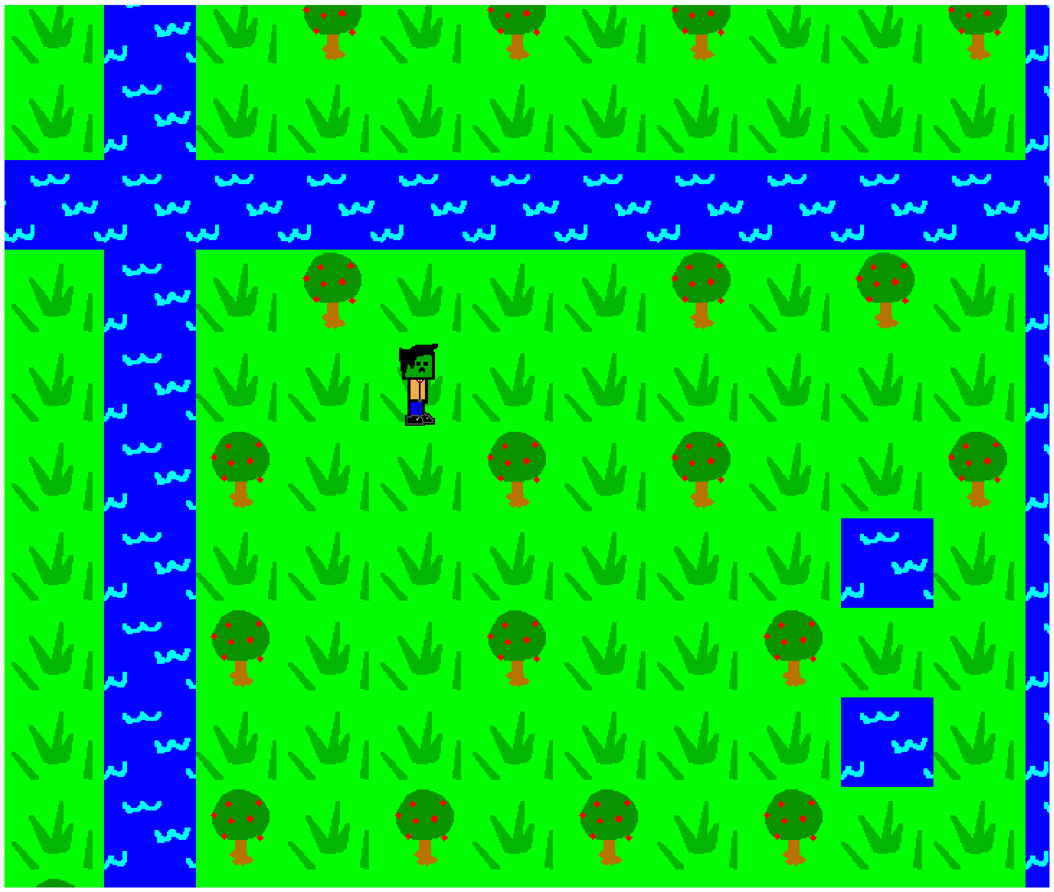
Project Overview:

For our project, “Deforestation Simulator,” we created a top-down version of Minecraft/Terraria (albeit with limited features when compared with the actual games themselves). The game procedurally generates a potentially-infinite world, in which the player may “move” - walking around harvesting and placing blocks.

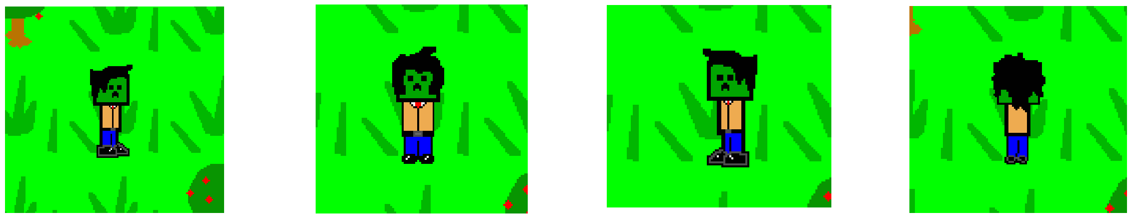
Results:

We have created a game with currently limited features, but with a fairly robust and open-ended framework which be adapted to a wide variety of different aims.

In our current iteration, the game has:

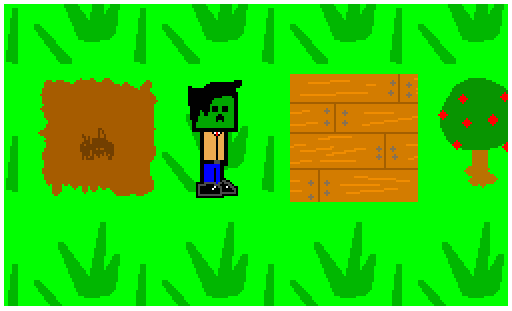


An open world, which procedurally generates grass, trees, and lakes.

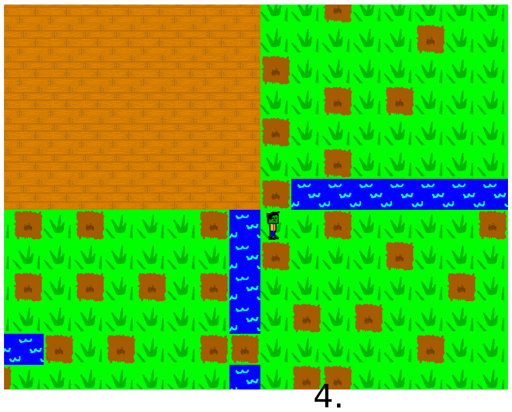


A player “avatar,” which can move in this world and changes appearance based on where the player is moving/facing.

1.

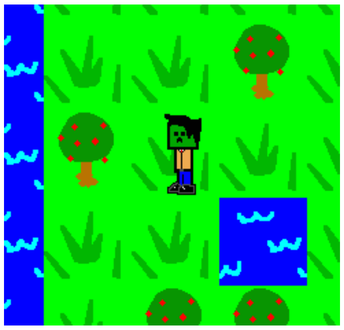


2.

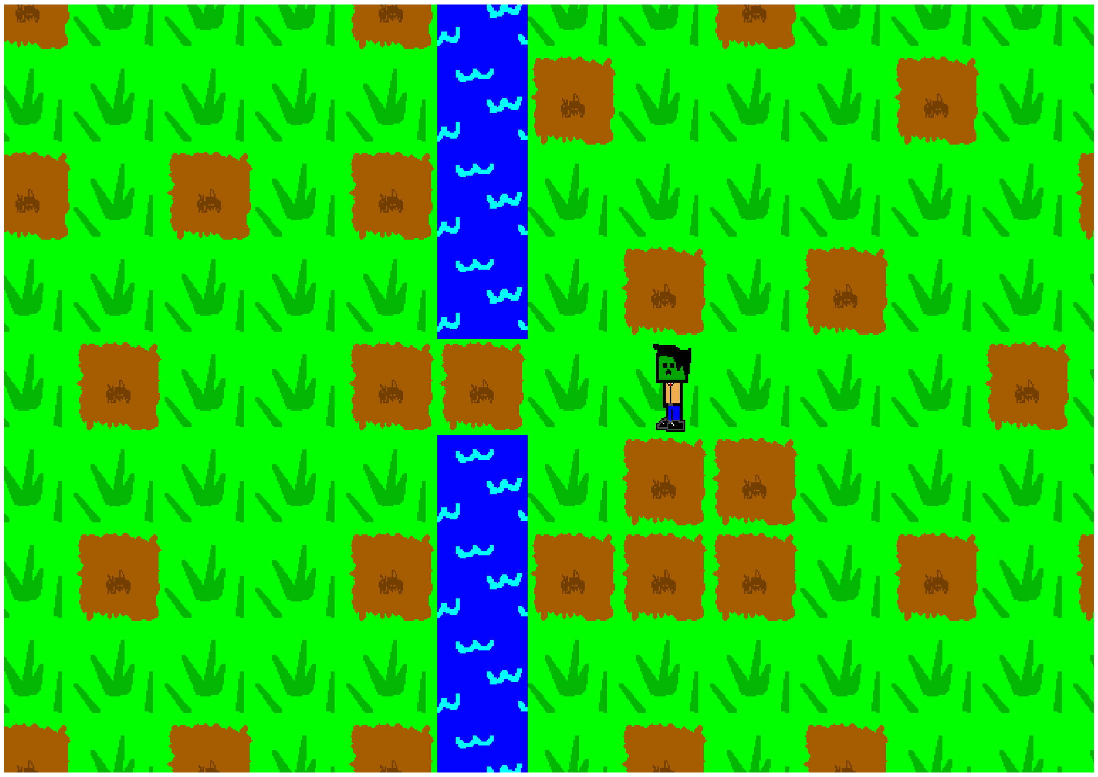


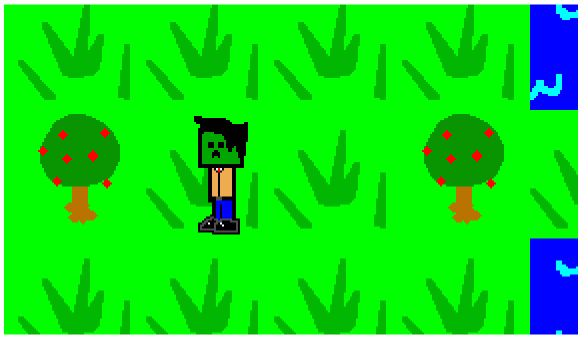
3. 4.

The ability to mine (“dig up”) trees, and then place an equivalent amount of wood into the world!

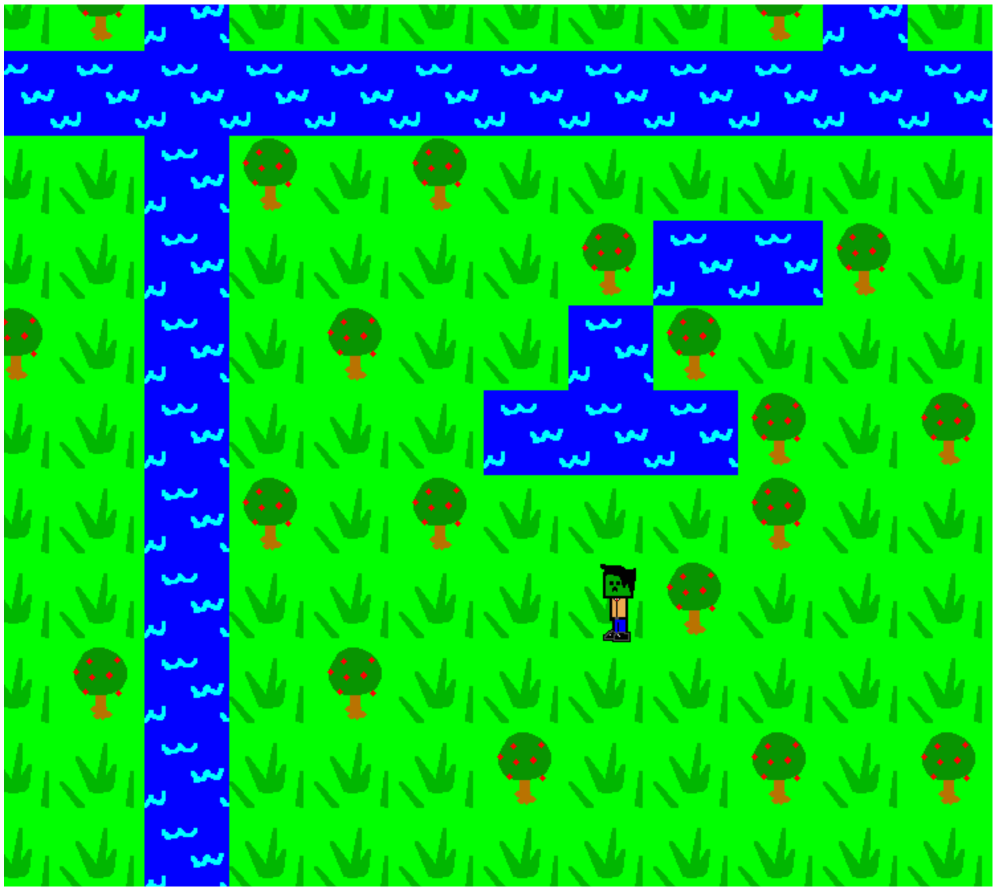


The ability to *not* walk through trees, wooden walls, or water, but *to* walk through grass and dirt!



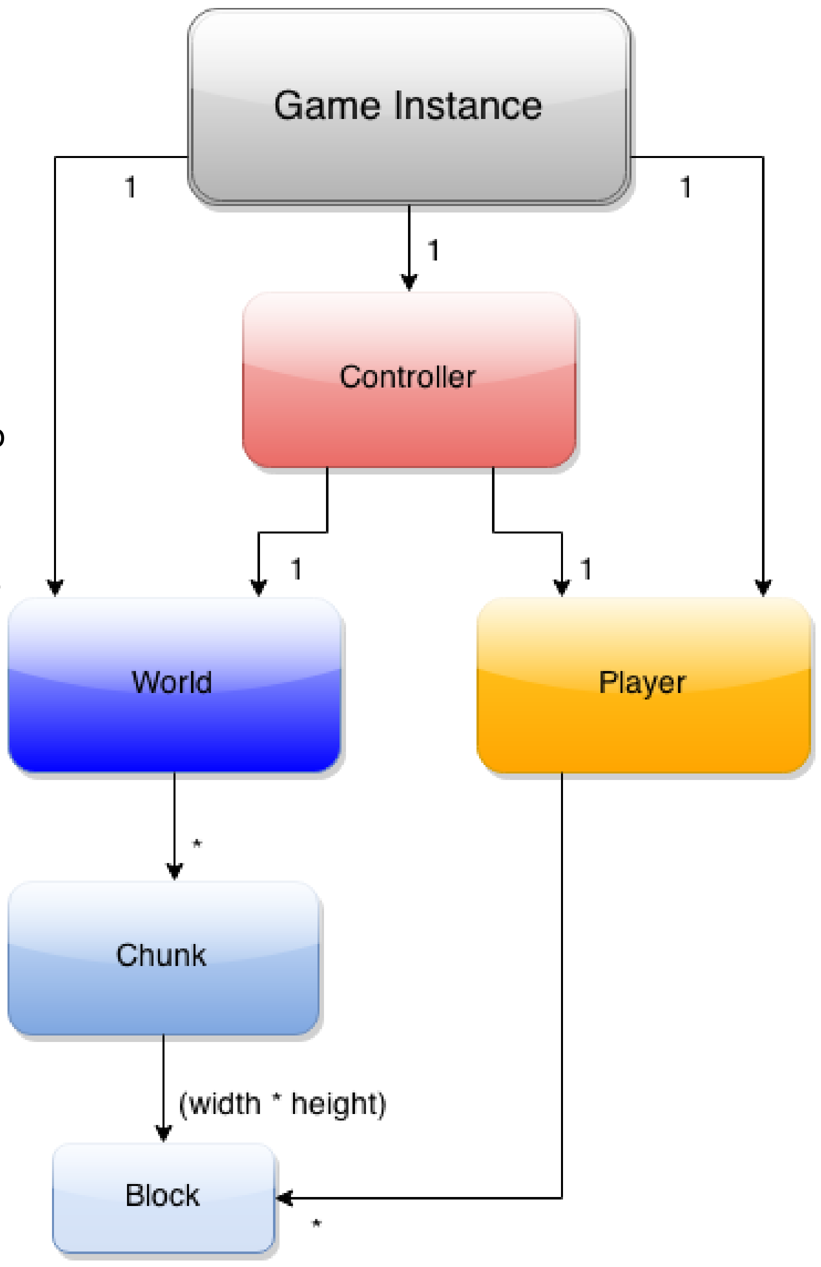


The ability to destroy trees and lakes in an act of industrial furor, only to discover that these natural resources can never be truly replaced—leaving you alone in a lifeless world with naught to do but contemplate what you've done...
...or, to continue walking in the *open world* to find new, uncharted wilds to exploit!



Implementation:

Our implementation consists of:



World:

A “world” object, initialized at the start of the game. The world creates a few “chunks” of blocks (see Block) at the beginning, with semi-random procedural generation (e.g., trees have a small chance of appearing, and a lower chance of appearing adjacent to each other. Water is very rare individually, but once one block appears others are more likely to generate around it. There are giant rivers going along the x- and y-directions, because we liked how they look.). The world is what the moving, mining, placing, etc. functions check, and is what we use to get the images to draw/blit.

As the player enters new areas, these new “chunks” are generated, and old regions are pickled in the chunks folder; if the player returns to a previously-visited area, instead of initializing a new area the game reloads the region like it was last left. By having it organized like this, instead of loading everything at once, its' easy both to save the world state, and make the world as big as we want it to be.

Block:

An object which consists of a common name, graphic, various interactions (e.g., whether or not the player can walk over it, whether it can be destroyed, what it gives the player if it's destroyed).

These blocks, *en masse*, comprise the world. Initially, we made the “tree,” “grass,” “water,” and other blocks their own subclasses—but, since every instance of a given block is identical (all trees are the same, grasses are the same, etc), we instead elected to make these defined functions which *returned* a new instance of the object, e.g. grass() returns a generic grass object, and tree() returns a generic tree object.

Player:

The player is an object which contains a few parameters—an inventory dictionary (which starts out empty), avatars, a position in the "world," and a direction. The player module also contains a few other parameters, like health—features which further iterations might expand upon, although they're not presently used.

The Player class has several sub-methods, which are called by the Controller (see Controller). These functions cause the player to "turn," move, harvest/mine blocks, place items (if they're in their inventory; otherwise, if they try to place something they don't have, it does nothing), etc.

Controller:

The Controller module is initialized to a "dictionary" (which maps various events, like keypresses, to actions), a Player, and a World. The class has the method process_events, which (while the game's "exit_flag" is False) checks pygame events, and carries out their corresponding actions for the controller's player.

At present, the game only has one player; as such, we initially just used a game loop (rather than a Controller class) to run the program. In the interests of both learning more about the Model-View-Controller model, and making it easier to change the control scheme (e.g., arrow keys vs WASD) or add extra players (e.g., initiate two different Controllers, each for different players), we added the Controller class; this had the added benefit of making the main "wrapper" file cleaner and easier to read.

Game Instance:

It is mostly used to separate the functions for processing and rendering the game. Its initialize function sets up everything in pygame, and also manages the random variables that change the size of the display window. It would also make it easier to run local multiplayer, if we chose to extend our program in that direction.

In addition to these classes and objects, we have a "chunks" folder which takes in pickled data from the game (so that the player can leave an area, then return to it later, without keeping progressively larger amounts of space loaded), and a graphics folder (which contains a variety of PNG files for the player's avatar, the background images, etc.), so as to keep the code files more organized and separated from pickled info or graphics.

NB due to the fact that our world is an "array" of blocks, and everything is graphics centered around the player, we elected not to create a "view" module or class. Simply blitting the graphic attributes at their corresponding locations was deemed simple enough that adding a whole new class & object would be unnecessary.

Reflection:

In terms of process, we consider this endeavor reasonably successful. Ultimately, our gameplay isn't the most engaging: although we've created quite a bit of architecture to expand, through adding more blocks or additional players, the core gameplay doesn't have much of an objective to it. That said, we both expanded our knowledge of classes, and learned how to implement the Model-View-Controller architecture (even though ultimately, our View wound up just being a couple commands in the wrapper file, rather than a separate part of the architecture).

Our project was reasonably scoped, more-or-less. Our original goals were actually slightly *under*-ambitious: our minimum viable product was reasonably quickly achieved, and could have been accomplished without learning much by way of software design. That said, our stretch goals gave us room to grow—and many of our stretch goals were a bit more than what we were able to achieve in the time we had (although this was substantially impacted by the fact that the last two weeks have been *very, very* hectic for Matt's Engineering for Humanity and Sean's MSMechanics, resulting in us both putting a bit less time into this project than we would have in the absence of these other obligations).

In terms of unit testing: due to the rather interdependent nature of the classes and objects (e.g., a world means nothing without blocks, the player needs a world to interact with, the controller needs a player to do anything), doc tests and simple unit tests proved impractical. However, our development process did incorporate testing and incremental development: we started out with very basic features (a non-infinite/closed world of grass, and a player which could just move around), and then expanded outwards. We further divided our code into modules, so as to make expansion easier and the final "wrapper" file clearer; as we made changes to the modules, we would regularly try out the main file to check if everything was still working. If not, we would create functions to examine the state of the code. In one case, the displayed world returned all trees while the world itself was still correct, which made it obvious where the error was occurring.

Other than scheduling, our teamwork didn't have any real issues. We divided our work (each tackling specific modules or functions), meeting or pair-programming to mesh them together or when an issue/bug/problem required both of our attentions. Next time, a more scheduled approach (rather than just agreeing to meet each other in the lounge in an hour, or to touch base with each other "some time today") would no doubt improve the process – although even without this scheduling, we still feel that the endeavor was reasonably successful.