Depths of the Mind

By Connor McCloskey

For my final project, I wanted to take a few of the items we learned over the course of the semester and synthesize them into one project. I also knew that, from a gameplay perspective, I wanted to focus on something without combat – as much as I love a good combat system, it’s so widespread in games that I thought it might be interesting to do something nonviolent.

To do so, I looked at a game that I had recently played and could take some inspiration from, *Gris*. This is a non-violent scrolling platform that challenges the players through puzzles, a number of which are centered on environmental hazards. I also looked at games with simple mechanics and artwork, such as *Thomas Was Alone.* Given my love of narrative, I knew I also wanted to incorporate some sort of traditional storytelling element.

To accomplish this, I looked at the examples provided by our textbook. Seeing that we were provided an example of a side-scrolling platform, my goal became to take that framework and add to it.

I decided early on that *Depths of the Mind* would be a story set in a VR-inspired environment. The main character would be a father in an accident-induced coma. An experimental treatment would help you overcome the condition – a VR interface that let the character escape the coma by collecting memories. These memories would be stories of things that happened around the main character while they were out cold – the unfolding drama of your family, with whom you have a strained and degrading relationship, their medical decisions on what to do with you, random visitors, and other opportunities for character development. Pulling from *Thomas Was Alone*, this let me create a block, circuitry-inspired setting that was doable with my own artistic level. It also allowed me to bring a narrative bent to the project.

To create a working demo of this concept, I made the following changes and additions to the side-scrolling platform framework from our textbook. Please note in my code that I have labelled all new areas as well:

* I broke out a few different classes into their own separate files and created new files where appropriate.
* I implemented a save file function. This is pretty rudimentary, but it works by storing the player’s current level in a text file. At the main menu, the player can choose to either load the level in the file or start a new game (overwriting the save file to do so).
* I implemented respawn points and a “death” system.
  + The death system (or in terms of the narrative, rebooting the level if the main character fails in their attempt to break out of the coma) detects if the player’s Y coordinate dips below the screen’s set height. If that happens, the game sets the player’s coordinates to the coordinates of the respawn point. I also removed the original sample’s code treating the bottom of the screen as a floor. One can also see that if you wanted to create a “lives” mechanic following this, you could easily do so, though that is another overplayed element in gaming that I wished to avoid.
  + The respawn points are also complicated by the fact that I wanted to have multiple – at least two, in fact. One for the first half of a level and one for the midpoint. To accomplished this, the game using another class called a Respawn Controller. This is essentially a placeholder/trigger volume in the world that the player is forced to collide with as they normally progress through the level. On contact, the Respawn Controller sets the respawn point’s coordinates to its own, effectively moving the point to where I want it to be.
* I created a more robust system for level creation. In the original iteration provided by the book, the platforms that made up the level were stored in each level class as an array of lists containing the platform’s desired coordinates and dimensions. This was then looped through to create platforms and populate the level’s list of them. This is effective if you’re okay with having zero artwork, which wasn’t the case for me. So, I created new classes for each type of platform and art piece I wanted. These would have set dimensions to accommodate art. This way, I now had Lego pieces I could build levels with. All I needed to do was have a list of coordinates for each type of platform I wanted, and in a very similar way to the original, loop through that list to add the platform to the level at the desired coordinate.
* I managed to create the wind mechanic! After a few early experiments involving adding negative movement to the player’s X axis, I realized that the game kind of already had something I could use – the calc\_grav function in the player class. All I really had to do, it turns out, was copy this, but apply the “gravity” to the X axis (adding a negative value, of course).
  + Controlling this mechanic was tricky, however. I knew I didn’t want it going 24/7 – that’s no fun! The player would have to hit trigger volumes to both start and stop the wind. It also wouldn’t be a playable platformer if you were unable to control your character because the wind was blowing constantly. So, I ended up creating four other tools to track and control the mechanic:
    - A trigger volume for when the player entered a windy area
    - A trigger volume for when the player exited the windy area
    - A timer for controlling how long the wind should blow (wind\_tick)
    - A timer between each gust of wind (wind\_timer)
* I added lines of code to player and level classes to handle collision for my trigger volumes for wind, respawning, and level exits. This was straightforward since the original already had a great mechanic of tracking collision via sprite groups. Everything I created was a sprite anyway (that is the basic building block of Pygame, after all), so I copied this functionality.
* How the player transitions between levels was completely redone. In the original version of this game provided by the book, the player would automatically jump to the next level after traveling a certain number of pixels. Now, the player must get to the end of the level and enter an exit – a sprite that they collide with as mentioned above – to trigger a cutscene, and after that, drop into the next level.
* Speaking of cutscenes – cutscenes! Sort of! These are text entries, but they are “narrative interludes,” so cutscenes made for a convenient way of labeling these. At the start of a new games, and at the end of each level, the player is presented with text constituting the story of the game. These is done by calling a function stored in the cutscenes\_py file as passing in the current level number. It then opens the necessary text file with the narrative content and prints that information to the screen. This function has its own loop, much like the base while loop we use to create these Python games. This allows me to let the player read the content until they hit a key to continue, breaking the loop and returning it to the main function.
  + Separating the narrative content into its own file makes it easier to edit this content in the future as well.
* The game also features a main menu. This is a function called before the main game loop, and like the cutscenes, feature its own loop for its own controls and to, you know, function as a menu and not something that disappears in an instant! This is also where I put the instructions for the game.
* It should be noted that for both the main menu and the cutscenes, I discovered a separate Python library that made printing multiple lines to the screen with Pygame going much easier called Ptext. I’ve linked my source to it below.
* The game will now also quit if the player hits the escape key at any time. Just a nice little quality of life thing for this small project!
* Thanks to the book, I learned how to add music and sound effects. I’ve linked my sources to each sound effect below.
* All of the art is my own and created using Paint 3D.

Should I return to this project, I already know there’s a number of things I would like to do:

* Create “enemies” of a sort. I don’t want to add in combat in any way, but these can function as more hazards for the player to avoid.
* More environmental hazards representing the player character’s body rejecting the VR technology that’s trying to save it. Literal firewalls, debris, and water are a few ideas that come to mind.
* I would refine the wind mechanic. Right now the player has to tap the arrow keys to fight the wind, I would make an option or revise this to just hold the key.
* I would add in more “camera controls” to make a more cinematic experience. I experimented with a fading camera for a long, long time for this, but I just could not get it to work properly. I could get the screen to fade to black, but not from black to the game! Following the advice of a few tutorials online, this was done by creating another surface on top of the existing screen and creating a loop that would continuously increase its alpha value, creating a “fade to black”. I figured that doing the opposite would work, but alas, it was a buggy mess.
* I would probably make the camera “follow” the player along the Y axis too. This would be, I think, a simple addition. Just as the camera “follows” the player now by shifting the world along the X axis, you would just do the same along the Y. So, if I’m thinking about this correctly, the code is mostly there already, it just needs implementation.
* Of course, I would like to make the full thing! More story, more interesting ways to use the wind mechanic, more everything!

Sources:

Ptext - <https://github.com/cosmologicon/pygame-text>

Jump sound - <https://opengameart.org/content/platformer-jumping-sounds>

Wind – <https://opengameart.org/content/strong-wind-blowing>

Exit - <https://opengameart.org/content/windy-sound-2>

Death - <https://opengameart.org/content/8-bit-jump-1>

Music - <https://opengameart.org/content/rpg-ambient-3>