Bryson Rudolph

CPSC 6160

Final Project Report

1 May 2021

Introduction

My game, “Channels,” is a top-down, grid-based game centered around routing ships to their proper destinations and staying alive as long as possible. Players (there can be up to two) navigate along an island in the middle of the screen and modify the environment by using three tools they can freely cycle between, the shovel, bucket, and mop. The island itself is grid-based, but the players can move continuously and are not restricted to staying within tiles (i.e., the player objects can “straddle” two different tiles at once if they position themselves in a certain way).

The players share health points, which are displayed in the top left of the screen. Making mistakes will deduct health (which are discussed below), while successfully routing ships to their destination will add health. The final game score is simply the amount of time that passed from the start of the game until losing, meaning it’s paramount to keep your health as high as possible.

Game Mechanics

The most important component of this game centers around the ships, as they decide if health gets deducted or added. They come in four different colors, red, blue, green, and yellow, which correspond to each of the four sides of the screen (left side is red, right side is yellow, etc.). Each ship’s destination is one of these sides of the screen; specifically, it is the side that is the same color as the ship, and naturally, ships always spawn on a side that is not their

destination. However, ships are not intelligent and do not know how to navigate to their destination. As such, the players have to modify the environment to “guide” them, such as creating channels through the island itself. It is important to note that health is deducted if ships arrive at a side that is not their destination. The action of, in a sense, terraforming the island is carried out with the shovel and the bucket. The shovel removes tiles of sand while the bucket places sand. However, the tools have to be used multiple times in a row on the same tile to fully dig it up or fully place it down. In other words, sand tiles have different states of being removed/added. If the player uses the shovel once on a normal sand tile, it will be partially dug. If the shovel is used again, it will be dug to a higher degree, and the graphics will change to reflect that different state. The same goes for the bucket; using the bucket once will not replace the entire block to its normal, non-altered state. Currently, there are five different sand states.

All ships update on a fixed time step, which is currently ~eight seconds (and again, does not affect the players’ movement or abilities). Each time step, every ship on the screen will either move forwards in the direction they are facing or will rotate 90 degrees if a block of water is not in their path. The caveat here is that ships cannot backtrack. In other words, if a ship sees an obstacle in front of it, to the right of it, and to the left of it, it will explode (which removes health points). After exploding, the leftover hull will remain in the world and cannot be removed by players, which adds another level of difficulty as the game lasts longer and longer. Furthermore, during each time step there is a chance another ship will spawn in an empty space on one of the sides of the screen as long as the current number of ships on the screen has not exceeded the maximum (if the numShips < 50% of maxShips, the “chance” will be 100%).

An important mechanic deals with the last tool, with that being the mop. If a ship passes through the original boundaries of the island, it will leave a trail of oil. By using the mop, the player can remove the oil, which is critical since health points are deducted if a ship has to pass through it (as they will not actively avoid it).

Challenges

For the most part, development went along relatively smoothly. However, there were certainly some aspects of the game that required a good deal of thought. For example, figuring out how to detect the correct tile for player objects when they use their tool was not trivial, as not only can players orient themselves such that they straddle more than one tile but they also have different facing directions. As such, I had to do a lot of testing to make sure the tile that was "chosen" by the player made sense, even with these edge cases.

Perhaps the biggest challenge was getting all ship mechanics working correctly and in a manner that felt fair, simply because there were a lot of considerations that had to be taken into account. For example, spawning in a ship requires multiple conditions to be met, from making sure currentShips < maxShips to making sure they don't spawn in their own destination location to making sure they don't spawn in a tile where a previous ship spawned the previous turn (else that could seem unfair to players). After spawning, the ships' behavior had to be figured out as well. I wanted ships to be "dumb," else the game would certainly be a lot easier and could be played in a way that I wasn't anticipating (e.g., if ships always turn correctly to get to their destination, what's stopping players from just removing the entire island save for a couple of blocks where they are standing and letting the ships navigate on their own?). At first, this may seem easy (as it didn't require implementing complex A.I.), but getting the ships to the right level of "dumbness" certainly took time to test to figure out what worked the best.

Another challenge that I faced much earlier on was just the general design of the game itself. At first, I started developing the game such that all ships were the same and all players had to do was route them to any of the existing sides. However, I quickly realized this design could be abused. Since ships will automatically turn (as long as they have an open spot on one of their sides, minus the spot behind them), players could literally do nothing and the ships would reach the island edge, turn, and continue until successfully reaching a side. In fact,

players were discouraged from doing an action that I really wanted them to do, which was building channels through the island through which ships could pass. It made no sense to dig these channels, however, since again ships turn on their own and ships passing through the island leave oil trails that have to be cleaned up.

There was also a mechanic that I originally planned to add to the game but I couldn't get it working. Currently, when a player completely digs up a sand tile, water is found underneath. I originally planned on making it such that digging up a sand tile that is surrounded on all sides by sand tiles would just make a dry hole in the ground. Then, if a player makes a long line of sand holes and digs up a final tile that is next to water, water would flow into the hole tiles, thus filling them up. I thought this could be fun, because you could have an intense scenario when a ship is moving along a channel and the water hasn't completely filled up the entire channel yet. I made a recursive algorithm to handle this logic (wherein on each new sand hole it checks around itself to see if a water tile is present, and then it follows the path of holes if one exists), but I couldn't get it properly working in time.

Future/Next Steps

Probably the biggest next step would be to balance the game. Currently, the game works decently well in regards to the time step and ship spawn rate logic, but it can certainly be improved, especially with the two player mode. The game does increase the max number of ships that can spawn in the two player mode to make it more difficult, but more testing will still be required as most of my testing was naturally in the single player mode.

Other next steps that I'd like to take are improving the graphics and animations, namely with the static-image ships, since each time they move they jump forward an entire tile. I think it'd certainly look much better if the ships moved forward at a continuous rate. Furthermore, the players do not have any animations other than an idle animation, so adding a run animation as well as specific animations for each tool would be beneficial. Adding particle systems would be

nice as well, such as when a ship explodes or when a ship successfully reaches its destination. I'd also like to make the ocean more alive, as currently it's just a flat static color.

On another note, the world map currently is the same each round. I would like to not only add more maps, but more randomness in the maps as well. For example, I think I'd like the basic tile types to remain unchanged with each new map that I manually create (meaning if a tile is sand or water), but I'd like to add randomness such that certain blocks of sand are already partially dug and oil is placed randomly across the map. I feel like that would increase the replayability quite a bit.

Optimizations can certainly be added as well. Currently, the world is represented as two 2D arrays. One array contains information regarding the specific tile, be that sand, water, a ship, etc. The other array contains information as to if that tile is covered in oil or clean. Naturally, having to traverse two 2D arrays is not optimal. The simplest solution that I could think of that I'd like to implement is storing everything in just one 2D array. In order to preserve information as to if the tile is dirty or not, I envision appending decimal values to each element in the array (e.g., .0 indicates the tile is clean, .1 indicates the tile has a little bit of oil, etc.). This modification would certainly improve performance.

Closing

In conclusion, my final project "Channels" is a functional C++/SDL game best experienced with two people that is chaotic, challenging, and requires on-your-feet thinking. It is based around terraforming an island to guide ships to their proper destination, as well as cleaning up the oily mess ships leave behind. Even though it works and feels at least somewhat balanced, it requires more testing to improve the experience (such as with ship spawn rates/timing). It could also use graphical/animation updates/additions, more maps, and performance optimizations.