Collin Pham, Matthew Griswold, Daniel Thomas

Overview of the game

Keystroke follows the story of Grant, a spacefaring human who has become separated from his craft. Grant finds himself somewhere in the cosmos with a malfunctioning space suit. Near him is a planet with weak gravity. To activate his suits super jump, Grant must now button mash his internal keyboard with the correct keystrokes. He finds that as his suit experiences more wear, the keystrokes required to activate a jump become more complicated. Placed into a world of moving obstacles and platforms, he has one goal — survival.

Grant must avoid all obstacles that come his way by utilizing platforms and jump actions. There are four different types of obstacles. The Mushroom and the Crab are x direction obstacles. They are difficult to avoid because they must either be jumped over using keystroke combinations, or glided over by platform. The Fireball is an x direction obstacle that spawns at Grant's y position. This obstacle is difficult to avoid because it requires the player change their y position to avoid. The Asteroid is an x-y direction obstacle that spawns at Grant's x position with both an x and y velocity. This obstacle is difficult to avoid because of its unpredictable movement. In this world there are also platforms. Platforms are small "planets" that move in the opposite direction of x direction obstacles. They serve to supplement the already walkable ground and ultimately help Grant avoid obstacles.

When creating this game, we knew we wanted to base player control on this idea of random button mash. This design decision seemed new and interesting, but also unknown. When we took a step back, we realized the adjectives we were using to macroscopically describe how we felt about our physical input method also could be used to describe how we felt about the idea of the cosmos and space. So, we drew a corollary between the two in efforts to make players see an environment that mimicked how they were most likely feeling about this new method of control. That is, we attempted to create a bridge between the players visual and physical experiences.

Once we knew our game was going to be based around space, we curated the experience to elicit curiosity and uncertainty, as discussed above. This manifested itself in the audio and image assets we chose. Speaking to images first, we created a custom background in photoshop to make the player feel as if they really were in the cosmos. We chose unorthodox images for our obstacles, like a crab and a mushroom, to continue to stress this idea of "unknown" and "uncertainty". Further image decisions like a spacesuit with a flag reinforced this end goal. When picking audio files, we wanted to get the player excited, but still have a sense of awe. This was most obvious in our background music choice. To pick this file we

browsed through many space type instrumentals until we found a file that literally made us feel excited and awed.

Design of the game

For the very beginning, we set out to make a game that experimented with new input methods for gaming. We also initially decided that we wanted to make our game deployable on computers, both laptop and desktop without any additional software to install or hardware to purchase. That decision left us with just the keyboard as a method for collecting inputs since laptops don't have mice and desktops don't have trackpads and we wanted our game to be equally playable on both. This also influenced our decision to write our game in javascript, while we used a python script to run a localhost for the game party, our game can easily be hosted on a remote server and distributed over the web to be played in browsers.

Having decided to create a new input method we wanted to keep the other aspects of the game familiar so that the player could focus on the new input. With that in mind we selected a side scroller with jump, left and right commands for our character. Our initial idea was to have the player type out on the keyboard exactly what he or she wanted Grant to do in order to avoid the obstacles and subsequently death. We planned to gradually not give the player the commands needed to survive so they had to remember them and incorporate some degree of natural language processing so commands other than exactly what the game recommended would be valid. However, we quickly discovered that typing out whole words such as jump, left, right and slide was quite hard due to the relatively long time it took to type a whole word. This meant that in order to make typing whole words possible we had to make game play extremely slow which was not fun and still really hard. In addition, it was hard to scale the difficulty. We tried eventually not telling the player what to type in order to avoid the obstacles but that made our game more of a memory game than a game focused on a new input method and neither us nor our play tests found it to be much fun. We realized that our initial idea of whole words was technologically feasible but not feasible in terms of game play. We then added random sequences of letters to the actual words to jump and quickly realized this was both more fun and a better way to design the input. We dropped all whole words and switched to all key mashing for the jump input while keeping left and right movement controlled by the arrow keys to allow the whole keyboard for jumping.

Almost immediately we realized that there was a hardware constraint in the form that standard keyboard are unable to accurately log simultaneous key presses. Instead, keys pressed at the same time will be logged in random order. We adjusted the code to allow for random order of the keys required. In order to do so we created a key queue and checked if the most recent key presses detected matched the keys the player was supposed to mash simultaneously. However this also let player “save” keys in the queue. While originally unintentionally, we found during testing that this was necessary for stages where the jump codes were three or four letters. We also felt this feature made it more fun and did not make it too easy in that a new jump code would require a new three keys. Additionally we did not write this in the instructions, hoping that it would become something of a “cheat” players would discover with some repetition of our game.

In the early iterations, the keys for the jump keycode were randomly generated. During testing we realized that the position of the keys on the actual keyboard had a significant effect on the difficulty of the game. Especially in cases of when the jump keycode was changing rapidly, jump keycodes with letters far apart from each other were more difficult than longer jump keycodes with the letters all close to each other. We then devised a method to take care of this. We started with a graph of a keyboard that included each key connected to each of its neighbors. This starting information allowed us to write an algorithm to populate a dictionary to map key combinations to a numerical distance. Using this mapping we moved from being able to pick just the length of the jump keycode to additionally choosing the maximum distance that could show up between keys. As the player progresses in the game, both the length of the jump keycode and the maximum distance between keys will increase. In our own personal testing and testing with our friends we found this to be the most practical input, that allowed us to appropriately scale the difficulty while remaining both fun and achievable for the player.

Having decided on a key mash for jumping we then turned our focus on the world in which our character would be doing the jumping. As mentioned in the game overview we felt a space them fit well with our central tenant of a new and uncertain input method controlling jumping. We wanted to add platforms so the jumping could accomplish more than just avoiding obstacles for the player. The design for our space world is meant to be both humorous and slightly strange, reinforcing the nature of our jump input method. Making the graphics turned out to be an unexpectedly fun part of making Keystroke. We created the galaxy background from scratch in Photoshop explicitly so that it could scroll without a noticeable kink in the graphics. The platforms are an edited NASA photo of saturn, and we tinted a mario mushroom dark red to make it seem “evil”, again hoping to create a sense that the world was slightly off given that the mushrooms are usually green and good. The asteroid was isolated out of a different NASA image. The character, crab and fireball are off the shelf sprites. We noticed some of our more clever friends were simply hopping from platform to platform to avoid all obstacles so we adjusted the asteroids and fireball to be aimed at the character at the time of spawn.

During user testing we focused on ensuring that players quickly grasped and understood the rules and objective of our game, with the exception of the key mashing to jump, so that they could focus on learning the key mashing. We felt it was necessary that the game started easy and progressively got harder once the player understood how to play. We focused on testing an initial state that, save the jumping input, needed little to no explanation. To mitigate confusion we found during user testing, we added the white strip of “ground” at the bottom of the game as some players thought leaving an platform would kill them. We also initialized the character on that ground instead of initially having him fall from the sky to reinforce the notion of a “ground” that the character could stand on. We also added a “Press * to jump” string under the score to help avoid confusion when people first played the game. With these relatively minor changes we found our test players were very able to figure out how to play. Another piece of feedback we got was when users were progressing in the game. As soon as an obstacle reaches the end of the screen, the jump input changes to a new combination. Users reported having a hard time tracking the change, as the only place that updated was near the top by the score. To fix this issue, we added a floating letter that spawns in the middle of the screen and moves along with the player.

Implementation of the game

Figure 1. Mushroom Obstacle

Figure 2. Crab Obstacle

Figure 3. Asteroid Obstacle

Figure 4. Fireball Obstacle

Figure 5. Platform

Figure 6. Spacesuit Sprite

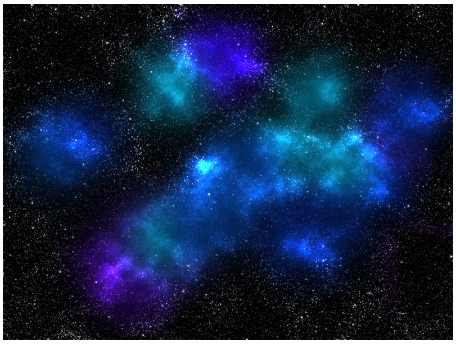

Figure 7. Background Image

We implemented Keystroke using Phaser.io, a javascript game engine. There were four distinct parts of our games that needed to be implemented — sprite control, obstacle handling, input recognition, and keystroke creation.

Sprite control was relatively simple to implement using Phaser. All we had to do was define our asset and then create conditionals for how velocity changed when arrow keys were pressed. Another movement that was relatively easy to program in was the jump functionality. A jump was executed after using the input recognition strategy mentioned below and checking to ensure the player had completed a previous jump.

Obstacle handling was also relatively simple using the robust Phaser physics engine. To implement obstacles, we made an obstacle class and held these objects in a global queue of current obstacles. The obstacle class stores the phaser obstacle object as well as attributes and information specific to each obstacle type. When it was time for an obstacle to spawn, we randomly selected an obstacle type from the four we had and used the obstacle class to feed the attributes to the physics engine which created it. This was then added to the queue of current obstacles. From there, obstacles were destroyed and removed from the queue if the player successfully avoided them. Platforms and floating letters are handled in similar ways.

We used an event listener to call a function for every key pressed and passed it the ascii key code. If the keycode was within the range we were interested in we converted the ascii key code to a char and appended it to a string. In our update function we would check if the last characters of that string contained the current jump keycode regardless of order, to allow for simultaneous key mashes. This key string was then cleared out after each successful jump to avoid double counting, if for example a letter showed up in two successive jump strings.

In order to create the jump keycode we used a random number function to generate random letters. If the game time indicated the jump keycode was to be more than one letter we then checked the physical distance of every subsequent randomly generated letter and if it was in range we would append it to the jumpstring. We would return the jump string when it reached the desired length. Both the key distance and the length for of the jumpstring were functions of the game time.

We struggled to find images in the public domain that fit the need for our background so we created a space galaxy background using photoshop. After using several filters and blending options we felt we had a pretty authentic image that was still able to scroll within the dimensions of our game without looking repetitive.

In terms of work distribution, we always coded as a group, tackling each of the above stated parts together.

Source code is included in the zip.

What went right

We are incredibly proud that users, regardless of levels of interest in gaming, found our game to be fun and wanted to play it multiple times. Watching our classmates come back again and again to beat their best score during the party was really awesome. They also beat the best scores we had ever achieved. We attribute this to the fact that while adding a new input method almost everyone was unfamiliar with, we paired with a side scroller that everyone found intuitive and familiar. So while trying to learn and then excel at the key mashing they did not have to learn the mechanics of the game. In addition we are extremely happy with how the graphics turned out. Initially they were something of an afterthought but after realizing things like a good background had an actual affect on people's' perception of the game we created graphics that added to the player's experience. Looking back at the project, pairing a completely new input method with a familiar game style turned out really well as people did not have to master too many things at once. We found that the key mashing was novel enough that novice and experienced gamers alike found our game to be enjoyable and fun while still challenging.

What went wrong

The largest failure we had was utilizing Phaser efficiently. Because we had never interacted with the framework, we ran into multiple bugs during implementation that could have saved us hours had we been more familiar. For example, when initially creating obstacles, our fireball asset was much larger than it actually appeared to the player. It took us a few hours to figure out how to correctly proportion the image because it was coming from a sprite sheet. We used photoshop to edit the actual .png and make it a layout the we could utilize phaser sprite control for.

We also had, and continue to have, trouble with our difficulty function. We knew we wanted the game to get more difficult with time, but creating a function to do so in a proportional manner was hard. We settled on an exponential that utilized exp(time), but still think the difficulty proportions are off. That is, the game is easy for too long, and then very quickly gets too hard.

One of our largest failures was with our mushroom asset. As discussed above, we chose to add a mushroom for user experience. We thought by changing the colors of it, everyone would understand it was a poison mushroom, and in fact this was reflected in user testing. However, during the final presentation, about 25% of people were confused by the mushroom and actively ran into it thinking it was something good. Upon reflection, we should have thought through our specific asset choices more and not underestimated degree to which mario is ingrained the memories of people.

Ideas for next time

In terms of debugging, we wish we had started user testing earlier and maybe even just bounced ideas off friends before Keystroke was in a testable stage. Many of the suggestions were very valid things that had never occurred to us as developers. Also we wish we had paid attention to graphics earlier, while developing the game we used stand in simple graphics to get the physics right but then had to retune

several objects at the very end when we swapped in our actual graphics due to dimensional changes and other unanticipated issues going from pngs to sprites.

If we were to continue development we would focus on additions that utilized the jumping mechanism we already created. We would want to explore making additional levels of the game considering we have already largely ironed the mechanics and physics of the game. In subsequent levels we would want to add coin like objects that could increase score. Additionally, these coins could be used to purchase skin packs and maybe even powerups from a store. Concerning platforms, we like the idea of making them self destruct so that the player has to keep jumping to new one. Now knowing that graphic design is incredibly important and fun, we would want to make new levels appear distinct by making new backgrounds, obstacles and even possibly new things for our character to have, for example a jet pack to help him jump higher with a special key input. Further, we would like to implement the ability for the player to do more than just jump. For example, the player could shoot a laser gun that destroys obstacles. We think this function would serve to key purposes. Firstly, it would make the game more interesting because the user could do more. Secondly, because there are now two actions, the player could have different keystroke combinations for the actions, which would add a new level of difficulty.

Overall, we were very pleased with the outcome of our game, but obviously recognize that given more time we could continue to add features that cause more engagement from users.