sjwelber mollyodonnell Final Game Design

Abstract

Our game is a puzzle game where the player uses basic block programming to navigate their character through a maze, where it has been trapped by a mythological villain. Playing as a folk hero of their choosing, the player's job is to assemble the moves their character will take, and then run their "program" to see if they are successful in completing the level. Levels will gradually increase in difficulty, and introduce programming mechanics such as loops, conditionals, and functions. The overarching goal of the game is to instill the real-world problem solving strategies that computational thinking affords. It aims to do this by teaching the player to write a program to control their character, as well as how to debug and refactor their code.

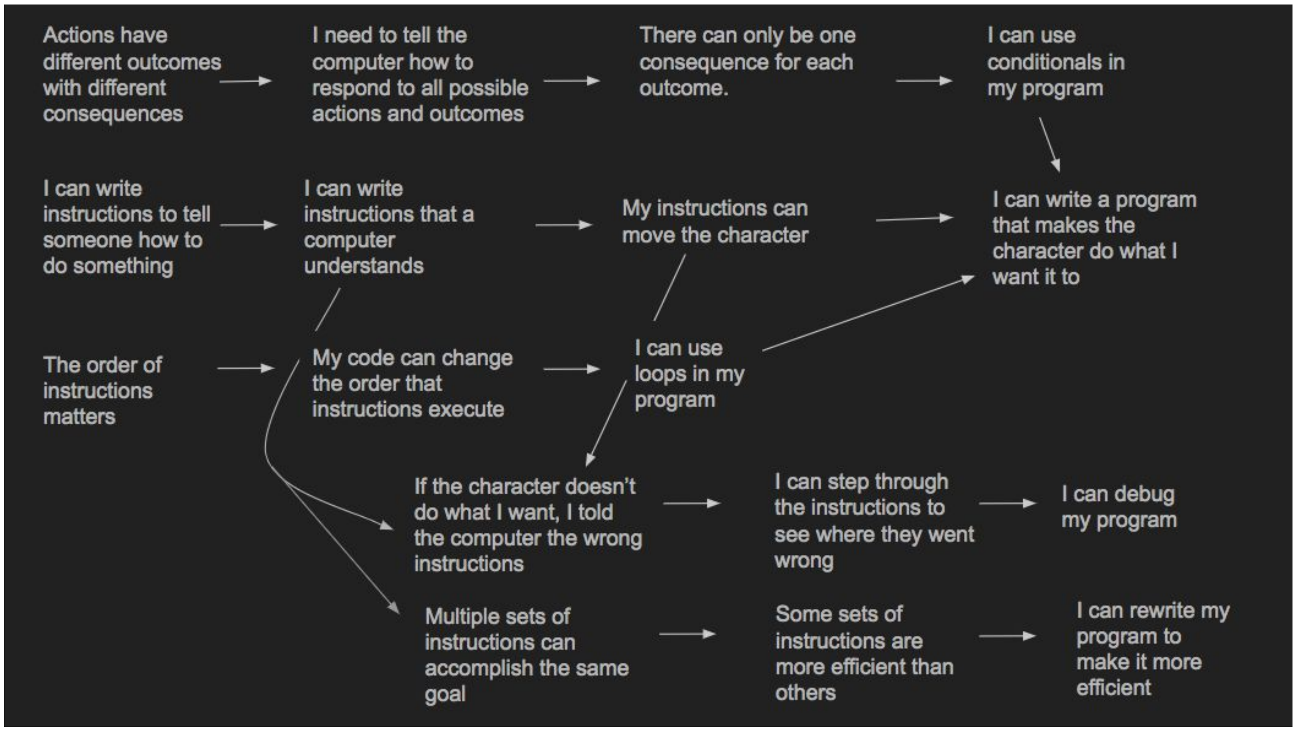
User Profile

We are designing our game for a user who is between the ages of 30 and 50 and whose first language is not English. Their reading abilities will be around a high school level or higher in their native language, but around a 5th grade level in English. Additionally, we are not assuming our users have math abilities above pre-algebra, although this is not very important to our game. We are planning for our game to be appropriate for all adults, but we designed with our most disadvantaged users in mind. However, the game will be targeted at users with no or little experience with computer science or programming and may not be as interesting to users who have CS experience. Our user may be more likely to have a smartphone than a computer due to socioeconomic limitations, so our ideal game is a mobile game.

Skills

We are going to teach basic computational thinking skills as they apply to real-world problem solving. By writing instructions to navigate a character through a maze, the user will learn how to break a problem into steps, how to come up with more efficient solutions to problems, and how to debug incorrect solutions. We will also teach more specific computer science concepts such as how to use loops and conditionals to solve problems. As the levels progress, these new concepts will be introduced and therefore the problem to be solved (the maze) will be more complex and will be harder to write steps for, restructure for efficiency, and debug.

Learning Trajectory


Actions have different outcomes with different consequences
I need to tell the computer how to respond to all possible actions and outcomes
There can only be one consequence for each outcome.
I can use conditionals in my program
I can write instructions to tell someone how to do something
I can write instructions that a computer understands
My instructions can move the character
I can write a program that makes the character do what I want it to
The order of instructions matters
My code can change the order that instructions execute
I can use loops in my program
If the character doesn't do what I want, I told the computer the wrong instructions
I can step through the instructions to see where they went wrong
I can debug my program
Multiple sets of instructions can accomplish the same goal
Some sets of instructions are more efficient than others
I can rewrite my program to make it more efficient

Our learning trajectory is partially based on the learning trajectories found here: http://everydaycomputing.org/public/visualization/.

Game Design

Our game is a puzzle game that is played on a smartphone. The game will consist of levels that get increasingly more difficult and introduce new mechanics as the game progresses.

The premise of our game is that you, a folk hero of your choosing, wake up trapped in a maze by the relevant cultural villain, with no idea of how you got there. The goal of the game is to escape the levels and eventually the location in which you are trapped by navigating through mazes. These mazes can have physical obstructions, enemies, and other features that you must navigate around or figure out how to move or otherwise alter. After completing ten levels, you will receive a new piece of information about where you are, why you are there, or something else relevant to the story. This story will be told through stills, animations, and music/sound effects, but not through spoken or written English.

The mechanic for controlling your movements is by programming them. Underneath the maze will be a set of empty blocks that take commands. You can select a command from a menu and drag it into the row of empty blocks. Once you are satisfied that your program is correct, you can run it and see how the character moves. As the program runs, the code block that is currently being executed will be highlighted. If your character successfully navigates the maze and reaches the goal, you complete the level. If not, the character will stop when it hits an obstruction, dies, or otherwise cannot execute your code, and the current block of the program will be highlighted.

At first, there will only be basic navigation commands available: turn clockwise 90 degrees and walk forward. The commands will be represented by icons, not words (up arrow, circular arrow). As the game progresses, more complex commands will be added to the options. These will include conditional commands that say "if ___ then ___", a frame that you can place around another command to have that command run a certain number of times, simulating a for loop, and a frame that says to run a command until a condition is met, simulating a while loop. Other potential commands that don't necessarily correlate with CS concepts will be: open door, close door, stab, hide, pick up object, use object, and put down object.

If you attempt to do something that cannot be done, for example, opening an already open door, the character will be confused and the program will fail. This is to teach the user how to anticipate possible errors and check for them. A correct implementation in this case would be to wrap the "open door" in a conditional of "if the door is not open".

There would also be an option to save a chunk of code as your own function. For example, if you need to turn left, you could insert three 90 degree clockwise turns, and then save these three commands as a new "turn left" command.

Skill Building

The user will learn CS concepts through creating the programs to move the character. Specific skills like loops and conditionals will be taught through program blocks that the user must use to navigate the maze. Debugging and the ideas of efficiency and refactoring will be taught implicitly as the user either makes and corrects errors or attempts to make the code more efficient to gain a better star rating for that level. The user will learn to break down problems into steps as they write the code to move their characters. Additionally, as the character navigates the maze, the code block that is running at that moment will be highlighted, so the user can see exactly how the code lines up with the motions.

Since our game is a puzzle game with distinct levels, the difficulty will get harder with each level. New specific CS concepts such as loops and conditionals will be introduced through new mechanics. For example, telling the program to "stab" will only cause the character to swing with the weapon once, so the user will have to either use a for loop (do 3 times: stab) or a while loop (until enemy is dead: stab). We will also give solutions a rating from one to three stars, so players can challenge themselves to go back and complete the levels more efficiently. As the levels progress, players will be expected to write more complex code, which will allow them utilize and build on the skills they have gained through playing.

Motivation

One way we will engage our users is through the premise and story. The premise gives players a motivation for wanting to navigate through the maze, and the story makes players want to complete levels to find out more. Another way we will engage users is through balancing skill level and difficulty to maintain a "flow" for the player. We will do this by increasing the difficulty of levels and adding new mechanics once they have mastered the old ones. Additionally, struggling players can still solve a level inefficiently and move on, while players who are advancing quickly can challenge themselves to improve their code and earn all three stars.

Our game appeals to many different player types. At its core, it is designed for the craftsman, as it is a puzzle game that requires figuring out the most efficient solutions. Your score for each level will be based on the number of commands you used - as each level will not have one discrete solution. This means you will be rewarded for efficiency, on a 1 to 3 star scale. This will also appeal to the competitor, the achiever, and the collector - you can compete against your friends or yourself as you try to achieve and collect all three stars on each level. There is also an option to create your own levels and share them with your friends, which would appeal to the director and the artist, as they can create new challenges for themselves and others. This would also appeal to the explorer because they would get to play through new mazes created by others and discover new ways of creating challenges in a maze. Additionally, all levels will have multiples solutions, so the explorer could enjoy figuring out different ways to progress. Finally, although the narrative won't be the main focus, it will be compelling enough for the storyteller to be able to connect with the main character.

Designing for the User

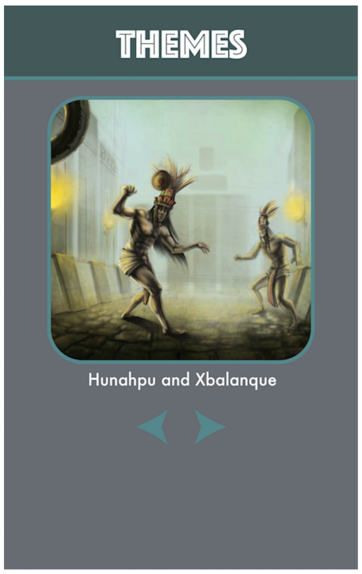
The visual, block programming style is catered to our target user, who has a limited grasp of english. One should be able to play the basic game and ingest the narrative without needing to read anything. The block programming style also simplifies programming concepts to an intuitive level, so those with no programming experience, such as our target users, will be able to start with no previous knowledge. Also, we are not introducing more complex programming concepts such as loops and conditionals until later in the game, and so the player will be able to focus on the initial goals of the learning trajectory, like breaking problems into steps, without being overwhelmed by technical concepts. Additionally, we are not assuming that our user wants to learn computer science skills, so the goal of our game is not tied to learning CS but rather to solving a puzzle and escaping a maze, but the mechanic "tricks" the user into learning the CS concepts.

Additionally, since our target user may be of a low socioeconomic class and so may not own a computer, our game is designed to be played on a smartphone. This affected design decisions such as layout, and level transitions. Also, since levels are self contained, and easy to replay/restart, players can take a break and come back to puzzles they are stuck on. This modular style is useful in mobile games, as it can be played during brief periods of down time during one's day.

Culturally-relevant Instruction

We designed a theme selection system to make our game culturally relevant. The narrative of our game is the same across themes - a hero is trapped by a villain, and they must escape, but the player will select a specific folk hero they would like to play as, which will determine the weapons, names, environment, features, and specific motivations of the character. For example, one of the heros is the Mayan twin Hunahpu, who will be rescuing his twin brother from the evil bird demon Vucub Caquix. Other potential heroes could include Gudit, a Jewish Ethiopian Queen; Sun Wukong, the Chinese Monkey King; and Badb, the Irish war goddess.

In this way, by plugging in different folktales to this format, we can include heros of all genders and backgrounds, and both target different cultures, as well as educate others about different cultural folklore.



Badge System

Achievement badges would be given for level completion, in increasing increments. (First for one level, then 5, then 15, etc.). Badges would also be received for earning increasing numbers of stars on levels, to encourage rewriting levels to be more efficient.

Participation badges would be given for various things to encourage persistence. For example, badges would be awarded for using “stab” a certain number of times, for failing a certain number of times, and for playing through on different themes.

Badges for academic success would be awarded for debugging and refactoring skills. For example, if a level was replayed for a different score, or if the player failed a level, cleared their code, and wrote it again, they could earn a badge.

Intelligent Systems

If we were to implement an intelligent system in our game, we would have it recognize when the user has failed repeatedly on a level that introduces a new block, or if they are not using that new block. Then, we would have a supplementary tutorial level for that block which first shows a video of the block being used, and then presents a level in which that is the only mechanic needed, other than moving forward.

Universal Design for Learning

To accommodate students with sight impairments, our game is not text-based, so being able to read text on a screen is not a requirement. However, we would also set the game up so that a screen reader could describe the components on the screen. We would also provide various colorblind modes that would alter the color scheme to fit the user’s needs. Finally, each theme would incorporate high contrast colors for the characters, obstacles, and background, so that these aspects can be seen more easily by all.

To accommodate students with hearing impairments, we would ensure that anything in the game that is indicated by audio will also have a visual component. For example, a character saying “huh” when trying to run incorrect code will also have question marks appear around its head so someone with a hearing impairment can still see what is going on.

To accommodate students with cognitive Impairments, again we have designed our game to not require reading of text. Additionally, we will include a menu where students can access previous levels and a tutorial mode that the player can return to at any time to relearn basic navigation. The game’s menus and interfaces will be logically oriented and the icons on the buttons will be simple images that clearly indicate what the button does. Finally, we will have a sandbox mode where users can create their own levels, which could be helpful for students who have trouble focusing on solving the provided puzzles or who just prefer creating their own mazes.

Tangibles, Movement & Collaboration

The unplugged activity that we would introduce, if this were being used as a system in a classroom, is an activity in which players direct the teacher. Players would construct their own physical maze with custom blocks, depending on which level they were on. Then, they would have to direct the teacher through the maze. As random elements are introduced, such as enemies, more participants can be involved, as they would take the role of the enemies. In this way, players could work in a pseudo-sandbox mode of the game, and practice different moves in different environments of their own creation.

Assessment & Transfer

Our first learning objective is: I can write instructions that make the character do what I want it to. We would assess this objective by providing the students with a new maze they have never seen but that uses mechanics they are familiar with, and have them write out the instructions that would be needed to solve the level. The real-world learning objective that this helps teach is the idea: I can solve a problem by breaking it into distinct steps.

Our second learning objective is: I can restructure my code to make it more efficient. We would assess this by showing students a maze they have never seen and an inefficient solution to the puzzle (i.e. it doesn't use loops when it could). We would ask the students to change the instructions to make the solution more efficient. The real-world learning objective tied to this goal is: I can find multiple solutions to a problem and determine which is the best one to use.

Our third learning objective is: I can debug my program. We would test students on this one by giving them a maze they have never seen and an incorrect solution to it, and would ask the students what the mistakes in the given solution were. The real-world learning objective connected with this goal is: If the solution to a problem I choose doesn't work, I can figure out what steps of the solution were incorrect, and then decide how to resolve the issue, whether by fixing the incorrect steps or by using a different solution.

A bridge activity between our first learning objective and its real-world counterpart would be to give the student a simple real world problem that they should know how to solve, and have them write out step-by-step instructions that someone else can follow to solve the problem.