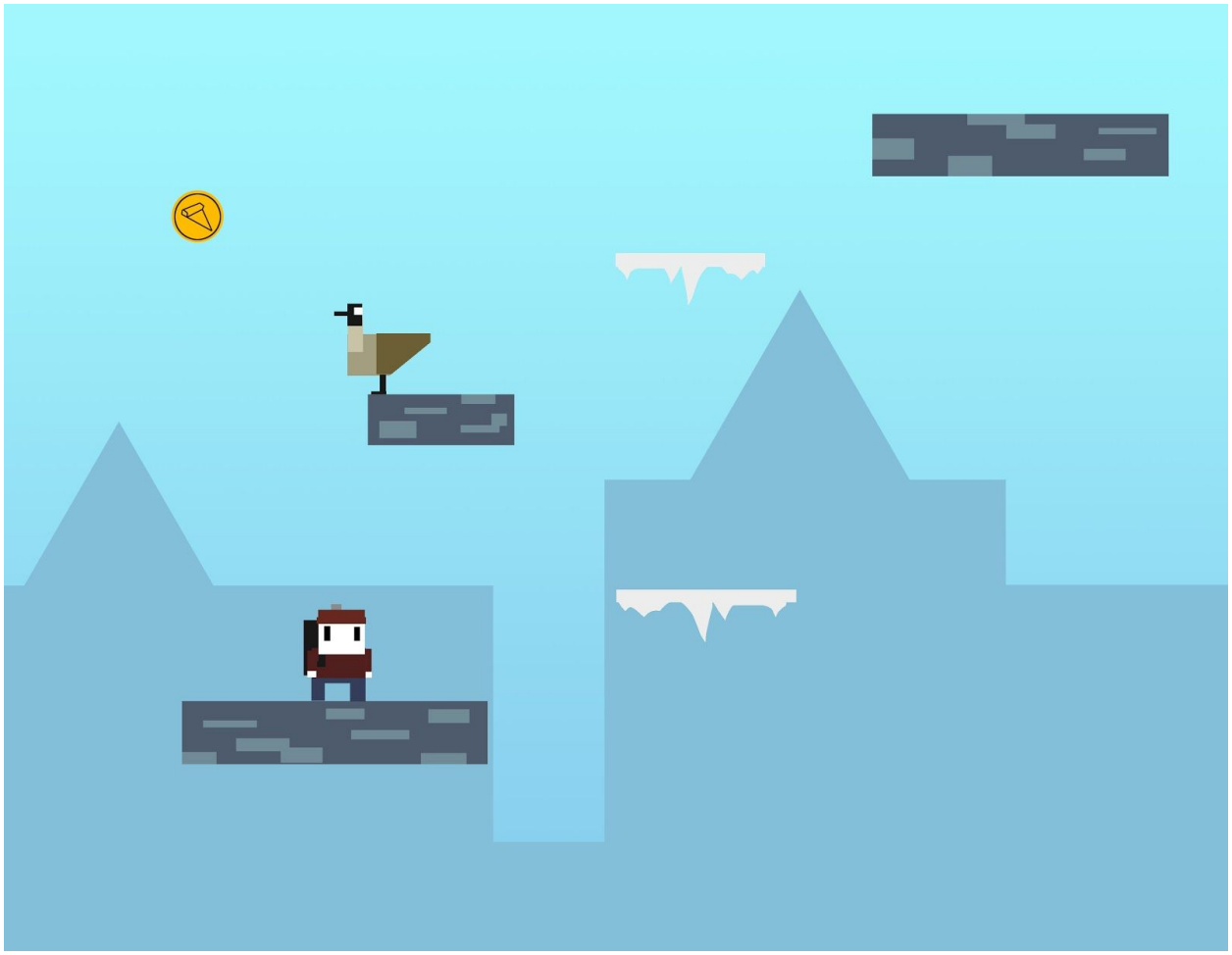
COSC 435 Final Project Proposal
Alex Rosenthal, Bryan Vaihinger, Tino Zinyama

Climbing the Hill



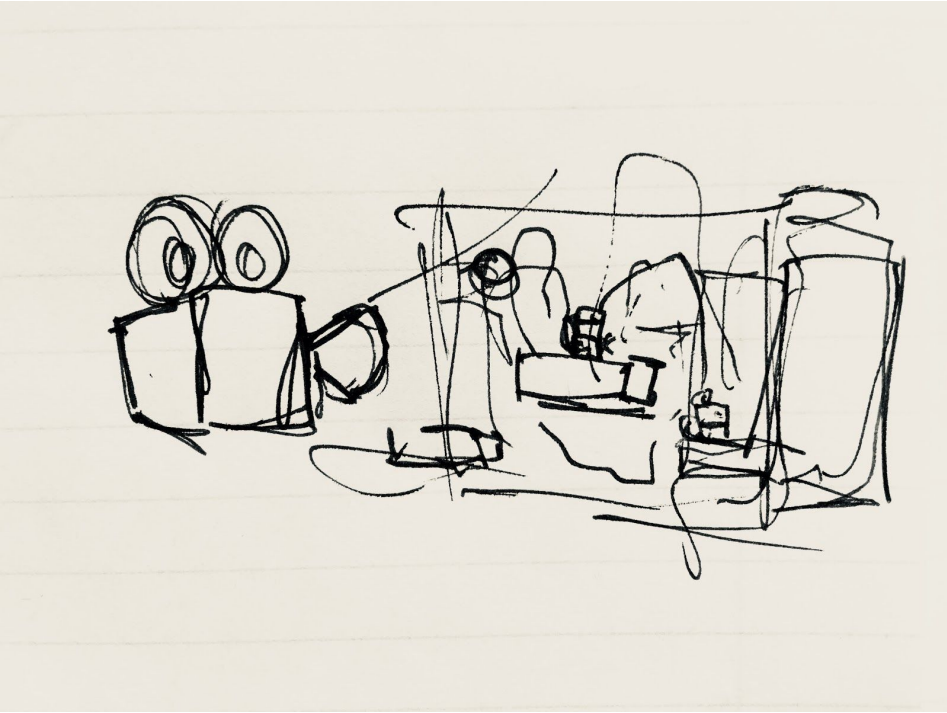
Statement

Our group's project is a platforming game based on the experience of climbing the hill to campus. The game will be mechanically 2D, but rendered using basic 3D forms to portray the platforms, hero, and gameplay elements. We have planned out multiple possible zones for this game to take place in, using different areas of campus as a backdrop. How many we manage to complete depends on how much time we end up having, but we plan on having at least one campus environment.

In addition to the scene, we plan on implementing multiple gameplay features to make the game fun to play. The fundamental mechanic of a platformer is movement, so satisfying lateral movement and jumping will be a priority of ours. This means responsive controls, variable jump height, horizontal lead-in and dampening, and gravity. Other interactive elements of each level will add hazards and objectives to the game. Some ideas we've come up with are:

- Goose enemies to avoid

- Stacks of books to climb or bounce on
- Falling ice platforms or icicles to avoid
- Strong winds that push the player, indicated by snow particles and “wind lines”
- Slippery ice that the player slides around on



We also plan on making a rudimentary level editing system to make the creation of multiple levels easier. Using ascii text to lay out items on a grid is a popular method that we think would be useful. Each zone will have a pre-made backdrop consisting of basic 3D shapes arranged to represent an area on campus, and then the actual interactive section of the level can be brought to the foreground.

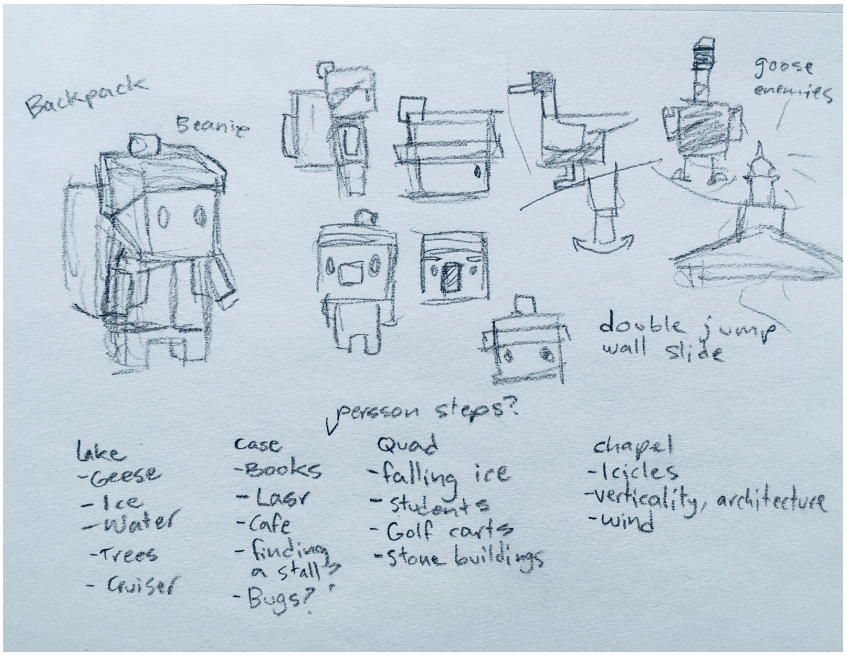
Overall, our ideas for our project are fairly ambitious. Keeping in mind that we only have about a month to complete everything, it may be that we have to settle for less than what we expected. With that in mind, we will build things incrementally such that we can stop at any time and have a resolved game that doesn’t feel unfinished or broken.

Technical Outline

1) In order for controls to feel responsive, the player model's position must not only process keyboard input quickly, but update and redraw the player model on the screen as many times possible in a second.
2) We will use a physics library, such as physijs.js, to help implement gravity, and interactions with gravity such as jumping and falling. We will need to animate the player model to perform some basic jumping motion, in order to provide the player with better visual feedback of their keypresses.
3) Collisions occur in the following general scenarios. The player object interacts with a horizontal platform by landing or standing on it. The player object is unable to move

through vertical walls. The player collides with an enemy such as a goose, or stage hazard like ice or icicles.The player object collides with some sort of bonus objective, such as a slices token. To appear polished the collisions must occur when the game objects on screen are in visually appropriate locations, e.g. touching.

4) A level editor allows us to much more quickly make levels, and the various zones within that the player can move between. The tutorial found on http://eloquentjavascript.net/ implements a basic editor that reads from a text file in order to add elements to a scene.
5) Use three.js built in library to handle the menu and options. The UI should provide the player with some sort of basic map to help keep track of their relative position within the zone, and display information such as score, time, and bonus objectives collected to the player.
6) Colgate's campus is not static, there are usually students, cars, birds, squirrels, and trees blowing in the wind. To some extent we want to capture that by providing some basic animation to background objects and scenery. We might take an approach similar to the aviator example and randomly generate clouds to move across the screen in the background.
7) We will use the three.js library's particle system to create the mentioned particle effects to give the game an increased sense of polish.
8) We will use a combination of a zone wide directional light to simulate the sun/background light, and smaller lights positioned within the scene for the rest.
9) Player model will be created from simple 3D shapes using hierarchical modelling. For animation, we can wiggle parts of the model to show running, jumping, etc. Below is a sketch showing a possible style for the player model.
10) Geese are notable, because unlike many stage hazards, they are mobile and can react to the players movement. A very simple state based AI will likely be implemented so that the geese can do simple things such as beginning and ending movement patterns. They will also be modeled and animated similarly to the player object.



Objectives

1. Satisfying movement
    a. Horizontal movement and air control that is tight and maneuverable
    b. Movement on the ground should be more sharp and quick than in the air
2. Jumping
    a. Variable jump height based on how long the player holds down the jump key
    b. Add a double jump functionality for more level design possibilities
3. Collisions
    a. The player can walk on surfaces
    b. The player can run up to walls and stop moving
    c. The player can interact with hazards or objectives
4. Level Editor
    a. Text based level editor that converts ASCII characters in a grid to rendered objects in a grid
    b. Option to add other elements not locked into a grid after text version
5. Basic UI & scoring
    a. Display the user's progress and score in start menu
    b. Should feature some information and options to restart, possible other settings
6. Dynamic background to make the world feel alive
    a. Moving background elements like students or birds
    b. Simple animations of things like leaves in the wind
7. Particle effects
    a. Snow particle effect in some of the levels
    b. Dust particle effect when running or landing from a high jump

8. Lighting
    a. Lighting might change through zones to simulate passing of time through the day
    b. Some game objects will emit light (buildings, cars, blue lights, etc.)
9. Player object model
    a. Player object should be simple but
    b. Style of the player object should match the rest of the game
    c. Player object will need animations for running, walking, jumping, etc.
10. Enemy Geese
    a. Geese will also need a basic model and animation
    b. Might have a very simple AI, so it will move in set patterns, or locations, given certain conditions. E.g. if a player moves to location X,Y, the goose begins moving to that position.
    c. If defeatable, possible have a particle effect to simulate bursting into a cloud of feathers

Bibliography

Platformer tutorial: https://www.gadgetdaily.xyz/build-a-3d-platform-game-with-three-js-pt1/

2D Platformer with Text Level Editor: https://eloquentjavascript.net/16_game.html

Basic platformer example: https://codepen.io/dissimulate/pen/CqIxk

Physijs, a physics plugin: http://chandlerprall.github.io/Physijs/

Using Physijs in a basic game: https://gamedevelopment.tutsplus.com/tutorials/creating-a-simple-3d-physics-game-using-threejs-and-physijs--cms-29453

Snow particle effect example: http://jsfiddle.net/3drjwj2n/

General three.js examples: https://threejs.org/examples/

Blender tutorials: https://www.blender.org/support/tutorials/

Using tween.js with three.js: http://learningthreejs.com/blog/2011/08/17/tweenjs-for-smooth-animation/

tween.js: https://github.com/tweenjs/tween.js/

Colgate flickr account for inspiration: https://www.flickr.com/photos/colgateuniversity/