Leonardo Ascenzi
Kerr Braza
Ryan Rios
Cindy Vo

Project Proposal: Raider Rumble

Statement

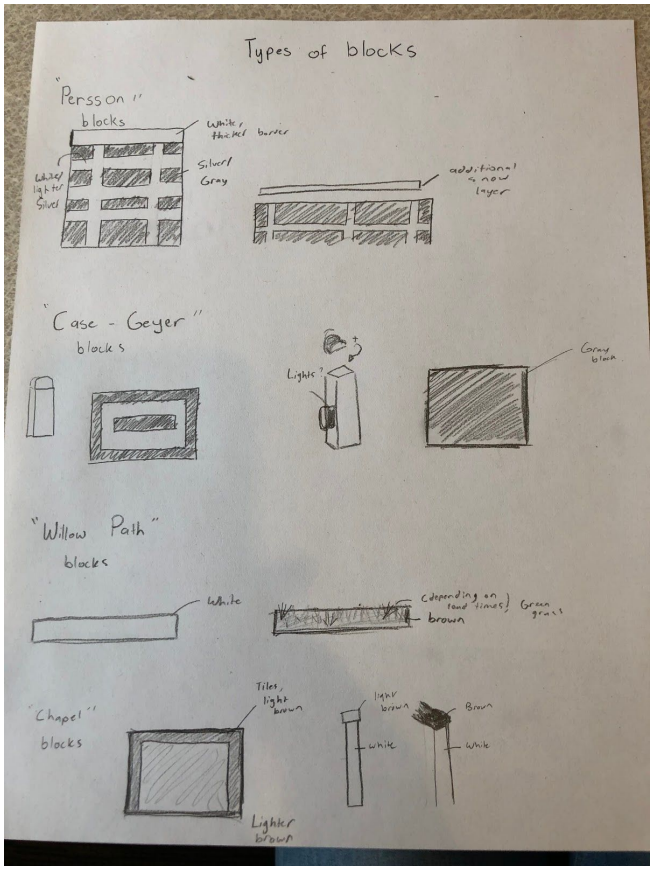
Our game is an abstract representation of those special instances when you meet a friend while walking around Colgate's campus and bond with them through silly interaction. We contextualize it with the Colgate's popular, scenic, and interactive hubs, where one is likely to encounter a friend or professor, such as on Persson steps or in front of the chapel in the academic quad. The stages will be abstracted through the animated-style blocks in the foreground and filtered pictures of the actual location as the background. On these stages, the characters then proceed to horse around with each other such as by giving them slice tokens (ranged attack), giving each other fist-bumps, or high-fiving one another (melee attacks). The game finally ends when a character is made to go out of bounds, signifying that they head off to where they need to be (such as the library or class). Part of what makes interactions special are the ephemeral property of their duration. The victory screens are thus the characters models waving in farewell.

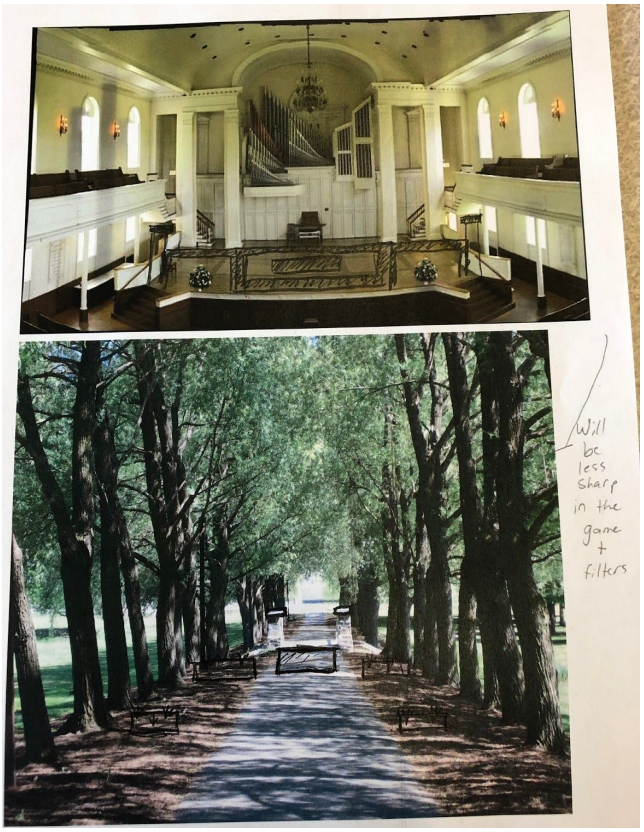
The focus of this game are listed in detail below, but our main mission is to create a fun game with fluid controls and have intuitive collision and physics that add to the happy and friendly nature of our theme: those special little moments with your friends around Colgate's unique campus.

Objectives

1. Collision
    a. Players can walk on platforms.
    b. Players being able to run into each other.
    c. Players can interact with each other.
        i. If they only run into each other, then they should collide into each other and stop, but they should not move back.
        ii. If one attacks the other, then the other should be knocked back.
        iii. If they both attack at the exact same time, then their attack animations should be cancelled out.
2. Interface
    a. There should be a UI element displaying the percentage of knock back.
    b. There should be a display representing the number of lives each character has. This drives the game loop and determines the end of a match.
    c. At the beginning of the game, there should be an introductory screen that says "Click to start" and two players have already loaded in and there will be an indication of which player is which character.

    d. At the end of the round, there will be screens indicating who won, and this determines the end of the current game session. Players will be able to play again by clicking on the screen with the same models.
3. Character Models
    a. We will first hard code two character models to keep things simple. Each character model will be different with different attack animations, appearances and style, and running/walking/jumping animations.
    b. The design of the models will be simplistic, which will be made up of blocks; therefore, it will have a blocky style similar to that of the Raider in *Climbing the Hill* game. They will also have a simple flat colour palette.
    c. Characters will have a "class" associated with them. These "classes" determine how much they are able to knock back another character.
    d. Our first two characters are Pingu and The Raider (sketches are provided in the **Sketches** section).
    e. Pingu is a penguin with a sword. It is of a neutral class, meaning that its knock back will be a base knock back (details of the classes and knock back are provided in the Collisions under the Technical Outline). Its basic attack will be having a sword swing out in simple arm rotation and translation movement in the left, right, and up direction. Its special attack is that it will dab vigorously and creates an area of effect damage around the model.
    f. The Raider will be a generic blocky character with a red raider cap, holding a slices token. Its class is heavy, meaning that its knock back will be greater than that of a character of the neutral class. Its basic attack is swinging slices token, which will be of the same animation style as the Pingu's sword swing. Its special attack is that it throws a clone of the slices token which flies straight in the direction thrown for two seconds or until colliding with an enemy.
4. Platform Models
    a. There will be platform models in the game that players will be on to fight or to knock each other off.
    b. They will be modelled after Colgate buildings, but will have a blocky style similar to the platforms in *Climbing the Hill* and *The Aviator*.
    c. Below are some example images:


Types of blocks
"Persson"
blocks
White, thicker border
Silver/ Gray
White/ light or Silver
additional = new layer
"Case - Geyer"
blocks
Light?
Gray block
"Willow Path"
blocks
White
(depending on round times) Green grass
brown
"Chapel"
blocks
Tiles, light brown
Lighter brown
light brown
Brown
White
White


Will be less sharp in the game + filters

5. Background/scene
    a. The background will be Colgate theme inspired, as it will be building(s) from Colgate's campus. The scene will also be generated through using ASCII codes.
6. Movement Animations
    a. Players can run, jump, and attack.
    b. Players flash white when they have been hit.
    c. Dust clouds to indicate jumping, double jump, landing, running, changing direction.
    d. We will have a max number of jumps set by a variable as in *Climbing the Hill*
7. Two players
    a. We will have the movements of the two character models be bound to different keyboard bindings so as to simulate the notion of two players.
8. Light and Shadow
    a. The type of scene will determine the lighting and shading for the stage.
9. Multiple Lives
    a. Each player will have a set number of lives. Once a player has been knocked off the stage that many times, they lose.
10. Loading Screen
    a. The loading screen will be used as a way to fill in screen time and have the user something to look at while the assets of the game are loading in.

Technical Outline

1. **Collision**: Collision and knockback physics
    a. Handle characters attacking each other
    b. Knockback scales with how much the character has been hit
    c. Design philosophy: Neutral, Heavy and Light classes exist which take different damage and fly at different speeds.
    d. Knockback will be a simple equation based on the vector from the center of one player to the center of another with a bit of Y increase so people fly up and back more than left and right.
        i. Multiplier for knockback speed is additive to current vector speed, uses current percentage, weight, and the knockback multiplier from an attack
        ii. Attacks fall into ranged, basic side and area of effect. Ranged and area of effect don't knock back very hard, but side attacks have the standard amount of force
    e. Jumping physics adds to only the y portion of the players movement vector and can be cast infinitely from the ground by pressing the up key for each player. In the air the y vector added is the same, but after that the player is locked out until they touch the ground.

f. Momentum is modeled by constantly dividing the vectors, for any direction, towards zero. This is only applied when the player isn't hitting any control. In the air momentum isn't slowed by the same factor
g. For checking if hits go through, simple distance check is applied with direction. If player1.mesh.clone().distance(player2.mesh.clone()) < model.width (normally a base of 10)
    i. If player1.facing right and player1.mesh.position.x - player2.mesh.position.x > 0, then apply velocity in -x direction etc
h. Gravity will be a constant vector applied to the -y direction

2. **Interface**: User Interface Elements
    a. Percentage display (representing knockback variable)
    b. "Stocks" representing leftover 'lives' of each character, determining end of match and game loop
    c. Victory screens for each character, stored in their object and demonstrating end of the current game session
3. **Character Models:** The characters
    a. Each character will be made of around 13 blocks. The main sections are legs, arms, torso and head. Stylizing block will also be added but not used for collision (such as the raiders hat, or pingus sword)
    b. The bounding box we use for collision encompass the main 10x20x10 area in which the player model is built.
4. **Platform Models**
    a. We will use Three.JS Object/3D meshes to generate these platforms.
        i. 10x10x10 hard blocks out of which to build stages
        ii. 22x2x10 floating platform types
5. **Background/scene:** Stage(s)
    a. We will first hardcode one stage that will contain platforms, background, and the characters
    b. Implement something similar to the *Climbing the Hill* model, transform Ascii codes into foreground on top of our stylized skybox
6. **Movement Animations**
    a. Jumping
        i. All characters will jump
        ii. Double jumping for more interactivity
        iii. Max number of jumps set by a variable as in *Climbing the Hill*
    b. Attacks
        i. As aforementioned, each character model will have different attack animations.

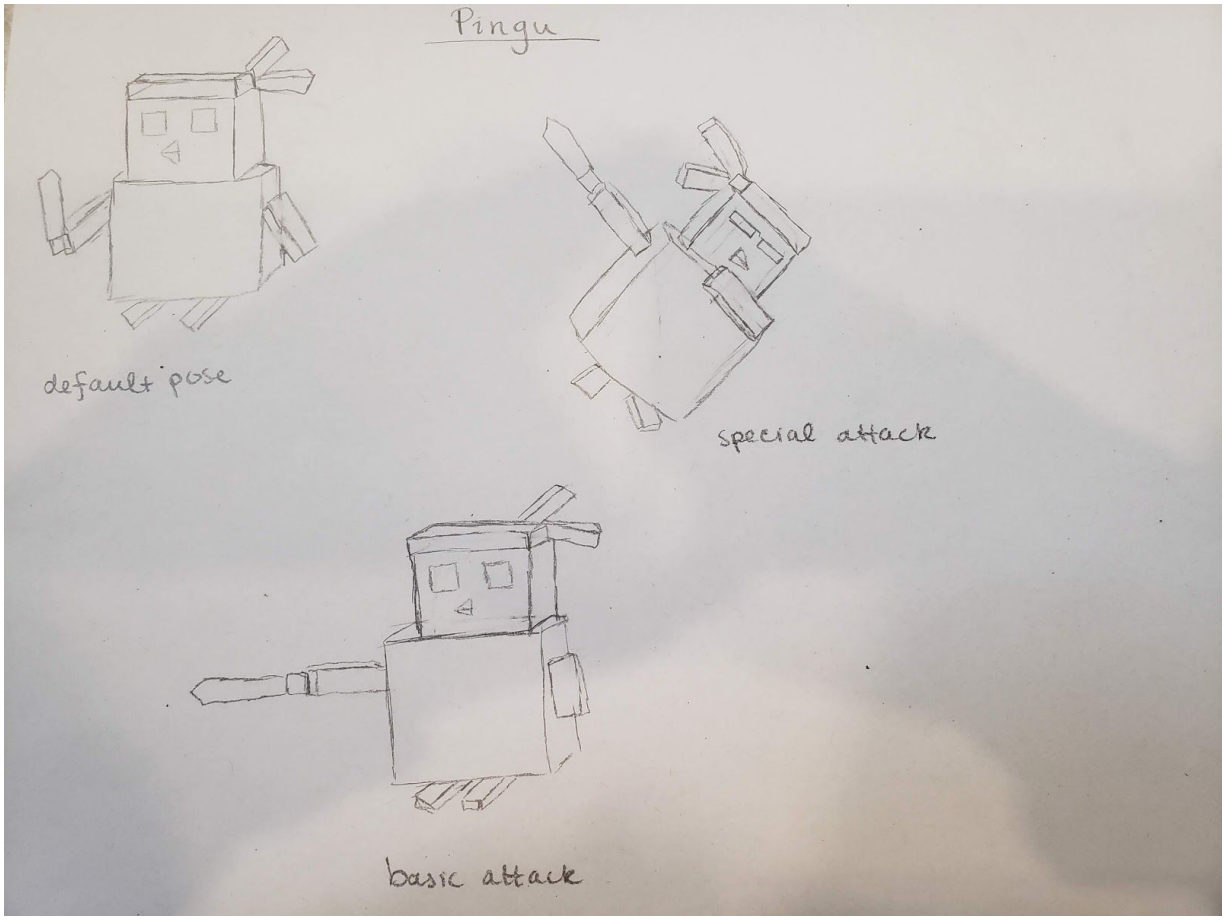
        ii. To make this more smooth attack animations, we will animate the movements incrementally so that the attacks will not look too jarring.
    c. *General Movement*
        i. Each model will have animations to move back and forth. Simple rotations on legs, arms and head.
7. **Two players:** Two players interacting
    a. The movement for each of the characters will have different keyboard bindings
    b. WASD, JK and ULDR arrows and 78 on the numpad will be used for up left right down movement, basic and special attack.
    c. Start off with 1 special and 1 quick attack that can be cast in any direction the player is facing.
    d. Implement jumping first, but double jumps shouldn't be hard
8. **Light and Shadow:** Particles and effects
    a. Have a directional light in a position and of a colour that makes sense for the map. For the Persson Hall Sunset map use a far away orange light emanating from the sun.
    b. Particles spawn on jumping, white blocky airclouds with random xyz rotation. Also on changing direction, small black particles spawning from feet opposite direction with random x rotation. White squares spawn from player hit when they are hit.
9. **Multiple Lives**: Respawning and losing
    a. Having variables that keep track of 'lives' for each of the characters in a game session which translates into ui icons of the characters and the end of the game loop as well as reset of the damage percent variable
10. **Loading Screens** : Stylized and representing Colgate
    a. Pause game loop in the background, waiting for assets to load properly
    b. Control with variables (stored in character object) which image pops up
    c. While the game scene is loading in the background, a foreground element to allow the player to feel like they're waiting less

Stretch Goals

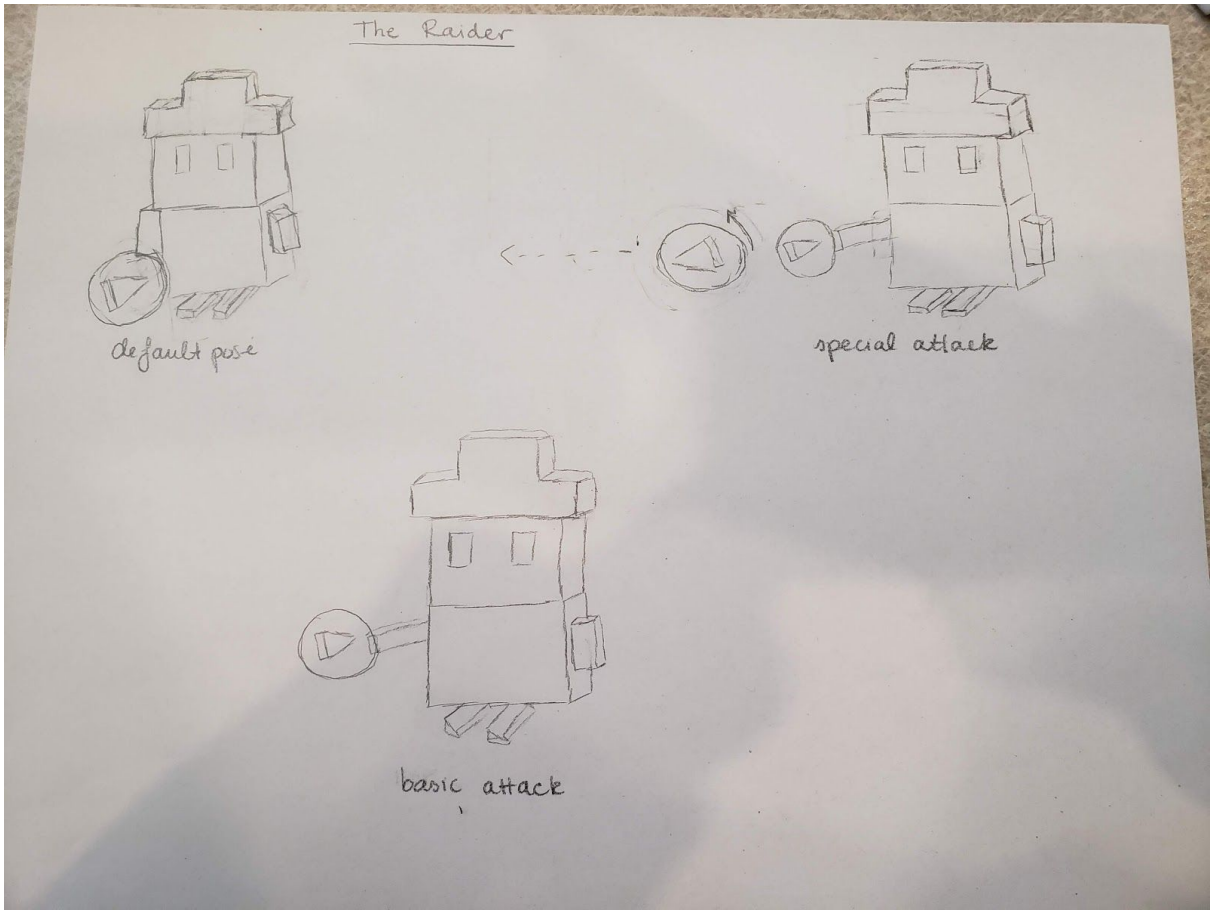
1. Controller support
    a. Nintendo Switch Joy-Cons
    b. Nintendo GameCube Controller
2. More Characters to Choose From
    a. Jo-Kerr
    b. LOD 4k
    c. On (Anh/Onh)
    d. Dynamic character making system

3. Platforms and Levels +
    a. Different block types, lava, ice, with different physics
    b. In game stage builder that will output a map to a text file and also allow it to be played on.
4. Camera Control
    a. Have the camera follow the players by calculating their centerpoint and leaving an appropriate amount of space on each side.
    b. Parallax and player tracking: vertical and horizontal
5. Grabbing
    a. Allow one player to grab another + throw them
6. Player summary
    a. See player damage at the end of every round
7. Choose from many stages on which to fight
    a. Before fighting, a player will be able to choose which stage to fight
8. Colgate based map selection using ray casting and a map of Colgate.
    a. Have camera focus on area surrounding cursor, click actual place on the google map of Colgate to fight at that area in-game. Selectable areas will have a very low detail 3d box model of the area on Colgate's campus.
9. Have a trail of particles follow player who's hit into the air
10. Have a trail of your character when hit hard, indicates a visual lock out of movement.
    a. gameLoop(){
    b. …...
        i. var tempTrail = player1.mesh.clone();
        ii. scene.add(player1);
        iii. setTimeout(function(){ scene.remove(player1)}, 100);
    c. …..
    d. }

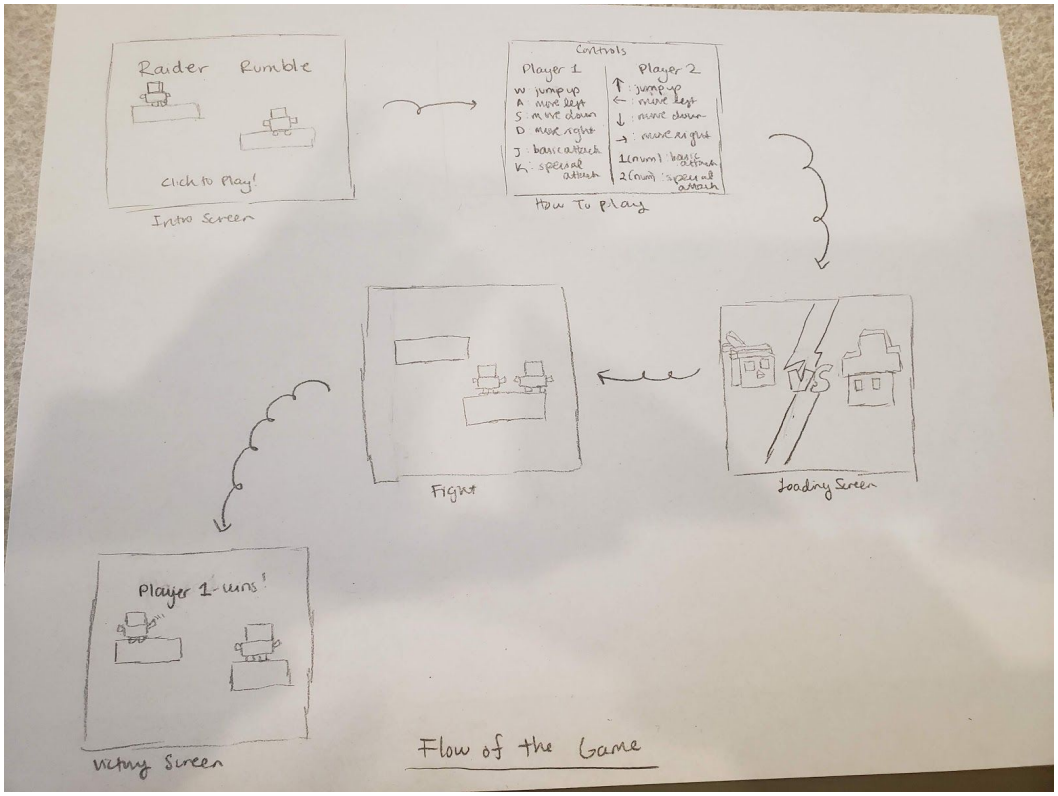
Sketches



This image shows the default and attack models for Pingu



This image shows the attack and default models for The Raider



This image shows the flow of the game

Bibliography

- Tutorials
  - http://stemkoski.github.io/Three.js/
  - https://developer.mozilla.org/en-US/docs/Games/Techniques/Controls_Gamepad_API
  - https://gamedevelopment.tutsplus.com/tutorials/creating-a-simple-3d-physics-game-using-threejs-and-physijs--cms-29453
  - https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene
  - https://tympanus.net/codrops/2016/04/26/the-aviator-animating-basic-3d-scene-threejs/
- Inspiration for game design
  - https://itch.io/games/made-with-threejs
  - https://www.colgate.edu/news/stories/13-best-colgate-images-so-far
  - https://gocolgateraiders.com/news/2010/7/21/FB_0721104908.aspx
  - https://www.smashbros.com/en_US/