CPSC 436D Proposal

Members:
Mark Pedersen, 37504140
Irene Fang, 38807129
Pinia Chandra, 22766142
Billy Liu, 36932135

Game Description:

After living a wholesome and fulfilling life, you have been re-summoned as a zombie to roam the Earth for eternity. Devouring rotten corpses, feeding on brains, hanging out with your undead friends, life was good. But all that changed when the Necromancers attacked. Using their black magic, they're spreading the H1Z1 virus which infects friendly zombies into minions of destruction under their control.

In order to escape the Necromancers' wrath, you must be in possession of the antidote once the brainwashing process begins at the end of the timer. However, you spot another zombie fighting for his freedom and there is only enough antidote for one of you.

Strewn across the dirt graveyard are brainwashed zombified body parts with which you may augment yourself. As a catch though, these body parts are under the influence of the Necromancers and are trying to find each other and become whole again. These minions of destruction are trying to stop anyone from escaping by giving the antidote to the necromancers to destroy.

To aid your quest for survival, there are randomly generated items that you may use to subdue your enemies. Also, attaching zombified arms allow more capacity to carry more items, and attaching zombified legs will increase your speed. Be wary that the more items you carry, the slower and more vulnerable that you are.

Once the timer reaches 0, the H1Z1 plague will spread across the graveyard. Friendly zombies without the antidote are brainwashed, and join the horde under the Necromancer's control.

Offensive Tools:

Missiles: Shoots toward opponent and creates an explosion. Opponent is blown backward some distance depending on distance from player and player's weight and loses limbs.
Rolling grenade: Bomb that detonates after a certain amount of time
Freezing: Stops the characters for a certain amount of time

Defensive Tools:

Reflection armor: reflects whatever hits it
Water: Creates mud that slows characters that step in it.

Passive Tools:
Arms: Attach to yourself to hold more items.
Legs: Attach it to yourself to run faster.

Keys:
There will two players, so there will be two sets of keys.
Player 1/Player 2 Keys:
UP/W: Move up
DOWN/S: Move down
LEFT/A: Move left
RIGHT/D: Move right
SHIFT/Q: Use item/Attack

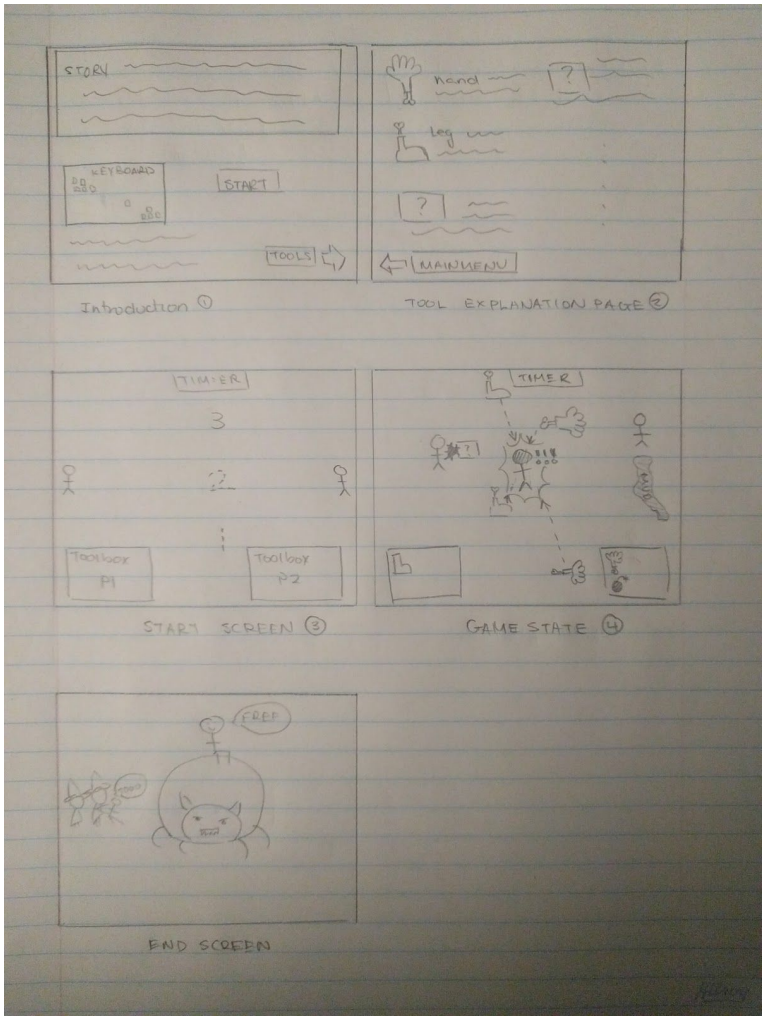
Rendering:
Game world, players sprites, special effects, limbs on ground, demon at end, and items must be rendered.

Geometric/Sprite/Assets:
All sprites will have their own two dimensional coordinate system. Sprites for zombies, limbs, and items will be made in Photoshop.

2D Transformations:
Character movement will be handled by matrix translations.
Different items will use matrix transformations. For example, the reflection armour as well as some item collisions with the walls will be implemented using a reflection matrix.
There are items that will also make use of a scaling matrix to enlarge/shrink the player.

Gameplay Logic:

- **Game States**



-

Concepts:

There will be 5 major game states (shown above)

- Introduction screen that explains the story and keyboard instructions
- Tool screen that explains the different tools in the game
- Start state with just the two main characters
- Game state when players start playing and tools start to appear
- Game state when there are 1 or more common enemy on the game screen
- Game over state where the timer runs out. Ends with an animation of the demon and the winner/loser before it goes back to the introduction screen.

- Player wants to be in possession of the antidote when the timer ends.
- Player can gain advantage by picking up arm limbs to gain slots to carry items
- Player can gain advantage by picking up leg limbs to walk faster

- Limbs gather together to form a common enemy that acts like a player and has the goal of also possessing the antidote.
- There can be multiple common enemies.
- Common enemies prioritizes getting the antidote, not attacking players. If a super zombie common enemy is in possession of an antidote, the other super zombie common enemies protect it.

AI related:
Resources:

- **http://drops.dagstuhl.de/opus/volltexte/2013/4333/pdf/4.pdf**
- **http://www.red3d.com/cwr/boids/**
- **https://arrow.dit.ie/cgi/viewcontent.cgi?article=1063&context=itbj**
- **https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc15/01Amitha/thesis.pdf**

- Missiles (AI)
    - Description:
        - Missile item randomly generated across the field
    - Objective:
        - Aim at nearest player
    - Implementation:
        - Pathfinding: A* search but will not be as effective against obstacles. The missile will move and turn slowly, so player can avoid it by intelligently avoiding it by making it crash into walls.
- Limbs (AI)
    - Description: Arms/legs will slowly come together to form a “common” enemy zombie. Randomly generated items that can be picked up to provide players with extra item slots or speed.
    - Objective: Come together (2 arms and 2 legs) to form a super zombie.
    - Implementation:
        - Pathfinding: A* search to find the shortest path to the nearest limb
        - Multi-agent pathfinding: Need to find the best and most efficient way to form a super zombie (combining A* search and strategy). For example, instead of joining with a lone limb closeby, both limbs would prioritize joining forces with a larger group of limbs that are slightly further but still nearby.
- Super zombie (common enemy AI)
    - Description: Formed from multiple limbs. There can be multiple instances of super zombie in one game.
    - Objective:
        - trying to take the antidote for itself
            - Finding where the antidote is

            - If antidote is owned by a player, attack player that is currently carrying it.
    - Implementation:
        - Pathfinding: Have one super zombie leader find the path (using A* search) to the antidote. This changes if that super zombie leader is eliminated.
        - Multi-agent pathfinding: for when new super zombies form. Implement flocking behavior AI: The rest of the zombies will follow that same path from their respective positions.
        - Decision tree: Priority of picking up antidote, not attacking players when antidote is unowned or owned by player. Priority of attacking players when antidote is owned by one of the super zombies.

Basic Physics:

- All gameplay elements have a weight associated with them. Elements with more weight will move slower and exhibit more force when colliding with other elements. Examples are included below.
    - Objects with more weight will move slower through some terrains such as water, mud. We can decrease the object's speed through these terrains as a function of the object's initial speed and their weight.
    - Projectiles will experience acceleration as a function w.r.t time and at the moment of impact we can use their velocity to calculate the force with which it hits the player.
    - An object moving at some velocity v hits a wall. The object will collide with the wall (an immovable object when compared to the object's mass) and bounce backwards. We can calculate the object's subsequent exit velocity by the formula 0.5m1(vf1)^2 + 0.5m2(vf2)^2 = 0.5m1(vi1)^2 where the right hand side is the kinetic energy transferred to the object, since the wall did not move at all (or a negligible amount). Final velocity will be vf1 = (m1vi1 - m2vi2)/m1. Since the wall receives the full effect of the force and repels an equal amount back to the object, the object bounces back at the same velocity if no energy is lost to external forces…
    - Projectiles will cause objects to be blown back a bit
        - vf1 = (m1vi1 - m2vi2)/m1
    - We can calculate the distance a body travels when hit by an explosion as follows:
        - Recognize the area of impact; for our purposes we could have the explosion happen in a circle and enumerate all of the objects within the circle.
        - If no bodies are within the area of impact, do nothing except rendering of explosion
        - Let p_c be the point at the centre of the explosion, p_i be the point of the bodies within the area where i = {1,2,3..n}
        - For each body, calculate the directional vector v_i = (p_c - p_i)
        - Normalize v_i to get v_ni = v_i/sqrt(x^2+y^2)
        - We can hardcode the explosive force generated and scale v_i by this number
        - New x, y coordinates of body will be f_e * v_i = <xi,yi>
        - Objects that have longer vectors will experience less force and will be blown back a smaller amount and experience less damage
            - We can do this by subtracting some amount from the force experienced at the centre for each pixel away from centre

Collision Handling:
Def: minimum bounding rectangle: MBR
We can find the MBR by following this algorithm:

1. Rotate the polygon onto the XY plane
2. Pick 1 edge and align this edge with the X axis(use arctan). Use the min/max x,y to find the bounding rectangle. Compute Area and store in list
3. Do the same for remaining edges in clipped polygon.
4. Pick the rectangle with the minimum Area.
5. Rotate Bounding Rectangle back for Coplanar reverse rotation of Step 1 and step 2

We will use a spatial hashmap (must implement the data structure ourselves, since not built in to c++) to limit the amount of collision checking that needs to be done. This way, instead of having to check every single object with every other object, we can define a subset of these objects that could possibly be colliding and only do collision checking on them. Since our game world is static, we can initialize a hashmap with (screen_height*screen_width)/2*avg_game_object many buckets with key and value (x,y) and a list of game objects respectively. On each update, we will delete their previous position from the spatial hashmap and add the game objects' current MBR's points to the map. All of our game objects will essentially be MBRs (can change later to more complex shapes), so we can hash all four of the box extrema and determine to which buckets they mapped. If boxes span more than one bucket, they will be added to more than one list. To query possible collisions, we can hash the game object's point and enumerate over all bucket values (possibly adding to a set to eliminate duplicates).

- Players can use direct contact or use tools like missiles and grenades which will cause the other player to lose limbs.
- Super zombies physical attacks on players will result in similar reaction to the tools (losing limbs) unless the character is holding the antidote, in which case the super zombie can attain it
- players attack on super zombie collisions will decrease the health of the super zombies
- Tools like missiles and bombs will have their collisions with walls handled by the physics portion
- When players collide with walls or other obstacles, they will bounce off of them with the same velocity but in the opposite direction, also further explained in the physics section
- Players collisions with tools will result in them picking up the tool, where the tool will go into their "toolbox"

Sound Effects:
Audio clips will be made using any free audio software. Sound clips will be made for the following parts of the game:

- Character death
- Reanimation of super zombie
- Acquiring items

- Attaching limbs to body
- Generic movement sound effect - sound to change depending on speed
- Item effects such as explosions, freezing effect, etc
- Collisions with each other as well as inanimate objects
- Background music: this will be a quiet chant at the beginning of the game that becomes increasingly louder as the game goes on
- Demon summoning at end of game

External Tools:
IDEs used will be according to preferences. Standard C++ libraries will be used. Nothing else known as of yet.

Bi Weekly Stages:
Week 1 & 2(Feb2) RENDERING WEEK:

- Setting up version control
- All the characters, tools and background
    - Characters
        - Rendered with texture
        - Basic movements with arrow keys and “WASD”
    - Background
        - Start screen
        - Pressing start brings you to the main playing page
        - Rendered with texture
        - Setting up grid map to help with pathfinding for later
    - Tools
        - Some with texture: water, arms, legs, freezing
    - Art
        - Player sprites
        - Background sprites

Roles:
Everyone: setting up version control
Billy: character movements and setting up grid map
Mark: rendering complex geometry tools and testing movements
Pinia:make game runnable and show graphics and background (help with rendering sprites and complex geometry tools)
Irene: design for art - creating the textures and assisting with rendering sprites

Week 3 & 4(Feb16) GAME LOGIC AND AI WEEK:

- Complete decision tree/pathfinding of limb, common enemy AI and tools
- Tools/items:
    - tool spawning
    - tool randomization
    - Freezing logic
    - Create complex geometry tools (missile and bomb)
- Usage of shaders for animations, using OpenGL 3.3 and GLSL
- Implement obj loader and renderfor .obj files (complex geometry tools)
- Implement gameplay
    - winning or losing condition
    - antidote random spawning in the beginning of the game
    - Timer implementation and render it on screen
    - Toolbox implementation and render on screen
        - Toolbox logic:
            - Adding tools (water and freeze) to toolbox slots
            - Adding slots to the toolbox when players pick up (collided) with arms
            - Using tools (water and freeze): decreases slots and the tool in the slot when attack keys (right shift and Q) are pressed by players 1 and 2. Tools used in first in first out order.
- Implementing game mechanics
    - Tools/items
        - Limb spawning
        - Limb grouping and pathfinding (Jump point search)
- Common Enemy
    - Art of common enemy
    - Common enemy movement mechanics
        - Pathfinding (A* search) to find player to attack and get antidote from, or pathfinding to find antidote.

Roles:
Billy: Create and implement complex geometry, loaders, and shaders
Mark: Implementing common enemy movements AI
Pinia: Implementing game mechanics limb AI
Irene: implementing tools/items and gameplay

Week 5 & 6(Mar2) COLLISIONS AND PHYSICS WEEK:

- Implement basic player actions:
    - Picking up body pieces if player “collides” with limbs
    - Implement manual attack collision (no items yet)
    - Dropping body pieces and items if hit
    - Dropping antidote if hit
    - Suit of reflective armor (reflects items if they collide with it), invulnerable for 1 or 2 seconds
- Missile collision
    - Missile pathfinding
- Implement collisions and environment physics
    - Implement walking/running physics:
    - Implement collisions with walls and obstacles: upon colliding with walls and obstacles on screen, players will bounce off of them
    - Implement projectile physics: recoil effect when colliding with players
    - Implement terrain slowing effects: upon walking on slowing terrain, players’ movement will decrease as a function of their mass and speed
    - RPG Launcher: implement function to calculate explosion effect on players within blast radius (refer to physics portion of document)
    - Rolling grenade: see above
- Work on more art assets
- Common enemy movement mechanics
    - Decision tree of whether they are chasing a player or antidote
    - Spawning - changing from groups of limbs to enemy
    - Multi-agent pathfinding: implementing flocking behavior for when multiple super zombies are on the game screen.
- Tools (all tools and limbs)
    - Limb/tool timer as limbs/tools collected can expire after some time
    - Spawning of tools and limbs within map boundary
        - Fine tune spawning positions so they do not spawn on top of each other and too close to each other
    - complex geometry: missiles, reflection armor, Rolling grenade
        - Shader implementations

Roles:
Billy: missile collisions, wall collisions, water physics
Mark: rolling grenade and missile and bomb explosion physics
Pinia: reflective armor collision
Irene: basic player actions and art assets

Week 4 TESTING AND EFFECTS WEEK (Mar16):

- Testing game to make sure basic game rules like terminating on time, collisions, physics and AI are working as expected
- Last winner state saved and transition back to start screen
- animations/effects
    - Player
        - Animation of movement(transformation depending on where it's going)
        - Rendering additional limbs when obtained
        - Rendering carrying the antidote when obtained
        - If a player is close enough to use “punching” attack or tool is very close to hitting, the trouble character starts flashing red
    - Environment Effects
        - Room lighting/mood
        - Fire/light (that appears for a certain amount of time) from the explosions of missiles or bombs upon impact
        - Water reflection/texture (for water in the environment)
    - Tools' animation/special effects
        - Flashing red of expiring tools
        - More advanced/improvement of the physics

Roles:
Billy: animation of player
Mark: testing and tools effects
Pinia: Environment effects
Irene: testing and transition to start page

Week 5 FINISHING TOUCHES WEEK (Apr14)

- Sound effects
    - Background
    - Explosions
    - Punching
    - Common enemy
- Tools page implementation
- Instructions page implementation
- Improvements on collisions and physics (if needed) and general finishing touches (can’t say for certain what these are until we are closer to the end of the game but leaving time for additional things that may come up during the game making process)

Roles:
Billy: tools
Mark: instruction

Pinia: sound
Irene: art for tools and instruction page