Minecraft
Due: March 6, 2018

Introduction

In this assignment you will build your own version of one of the most popular indie games ever: Minecraft. If you are not familiar with Minecraft, you can check out the classic version for free at http://www.minecraft.net/classic/play (requires Java). The game is based on a practically infinite block world in which players may mine (remove) almost any block. Players may also place mined blocks back into the world at any unoccupied location. During the daytime the world is peaceful, but at night mobs of enemies appear and attack the player.

The reason we are studying Minecraft is not because of its popularity, but instead because its underlying engine can be implemented in a matter of weeks. By the end of the assignment, you will have a fully playable Minecraft clone, complete with mining blocks, an infinite world, and simple enemies.

Demos of each checkpoint are available by running cs195u_demo minecraft{1,2,3}.

Support Files

Since you’re continuing to work with the same engine, there is no support code for this project. However, we have provided some assets that may be useful, which can be found at /course/cs195u/asgn/minecraft.

There, you’ll find a a 256x256 image named terrain.png. This is a texture atlas: a grid of 16x16 textures. You will be using this image to texture all the blocks in your game world. Additionally, there is a texture atlas to skin a pig, and 6 textures to make a minecraft-themed skybox. Feel free to use all or none of these assets: there are plenty of resources online with higher-res textures, different skyboxes, and more interesting NPC skins.

Week 1 - Due Feb. 13

Minecraft1 is all about setting up your basic world representation. The world will be divided into 32x32x32 chunks, each of which will be divided into 1x1x1 blocks. At the end of this week, you will have a small procedurally generated world.

Design Check

• How will you represent a block? What about a chunk?
• Explain how you will generate a Shape for each Chunk.
• Explain how you will texture your chunks.
• What will your procedural generation algorithm do? How will you use it to generate terrain?
Playtesting Requirements

- Handin only crashes on edge cases
- Movement and view controls must the same as Warmup
- Additionally, the world must have no gravity, so the player must have additional controls to change their height
- The world must consist of at least 1 chunk of size CxCxC, where C is at least 32

Primary Engine Requirements

- Handin meets all global requirements
- Blocks / Chunks
  - Block class, corresponding to a 1x1x1 cube with (minimally) the following characteristics
    - Transparency - whether or not the block should be rendered
    - Passability - whether or not a game object can pass through this block
  - Chunk Component
    - Holds an int / char array corresponding to the Chunks blocks (each int / char maps to a specific type of block)
    - The array contains CxCxC blocks, where C is a single parameters (or 3 different ones) determining chunk dimensions
  - Chunk Rendering (I)
    - A chunks blocks can be rendered

Primary Game Requirements

- Playtesting requirements must be fulfilled
- The world must consist of a 2x1x2 grid of size C = 32 chunks

Secondary Engine Requirements

- Handin meets all primary engine requirements
- Chunk Rendering (II)
  - Shape created corresponding to a Chunks blocks
    - The Shape contains only block faces that are adjacent to transparent blocks, with the exception of faces on chunk boundaries, which may always be present
  - Shape can be drawn
    - Shape stored in a drawable component
    - When the Shape is drawn, each face is textured based on the type of block that the face belongs to
  - Terrain textured using a single texture atlas
Secondary Game Requirements

- Handin meets all primary game requirements
- The terrain in the world must be procedurally generated
- Chunk composition is algorithmically determined with value noise (or something similar)
- Terrain contains at least two visually unique blocks besides air/empty blocks

Week 2 - Due Feb. 20

Minecraft2 focuses on adding collisions to your “voxel” world, but also features a number of rendering optimizations that will help you double your framerate while at the same time populating the world with 25 times as many chunks! Talk about fancy...

Design Check

- Describe the view frustum culling test.
- Draw and walk through a 2D sweep test (like the example from the slides).

Playtesting Requirements

- Handin only crashes on edge cases
- All Minecraft1 playtesting requirements must be met
- The game must run at 30+ FPS on department machines

Primary Engine Requirements

- Handin meets all global requirements
- Your engine supports view frustum culling of chunks

Primary Game Requirements

- Playtesting requirements must be fulfilled
- The world must consists of a 10x2x10 grid of size C = 32 chunks
Secondary Engine Requirements

- Handin meets all primary engine requirements
- VoxelSystem
  - Collision detection and sliding response for GameObjects with Chunks and Blocks within the VoxelSystem
  - Collision sweep algorithm implemented in X, Y, and Z directions

Secondary Game Requirements

- Handin meets all primary game requirements
- There is gravity
- Player is larger than a 1x1x1 block
- Player is able to collide with terrain using the VoxelSystem and the collision sweep algorithm

Week 3 - Due Feb. 27

Minecraft 3 adds raycasting to your “voxel” world for the purpose of interactivity. With this, the engine will be complete, and you’ll be able to add some real gameplay, and make your world nearly infinite. We’d like to remind you that even though your game may look like minecraft now, there is no reason you can’t make it a first person shooter, a racing game, or a puzzle game. Be creative!

Design Check

- Draw and walk through a 2D grid-based raycast (like the example from the slides).
- What is your criteria for determining if a chunk should be loaded or unloaded?
- When will you perform the actual loading or unloading of chunks?
- How will you implement addition and removal of blocks?
- What will your enemies be, and how will they be involved in your gameplay?

Playtesting Requirements

- Handin only crashes on edge cases
- All Minecraft 2 primary requirements must be met
Primary Engine Requirements

- Handin meets all global requirements
- VoxelSystem
  - Analytic voxel-based raycasting
  - Addition and removal of block methods
  - Addition or removal of blocks updates the Chunks shape

Primary Game Requirements

- Playtesting requirements must be fulfilled
- Addition and removal of blocks
  - Ray shot from the camera eye extending along the look vector
  - Visual highlighting of the nearest intersected face
  - Targeted block can be removed
  - Block can be added to the empty block adjacent to the targeted face

Secondary Engine Requirements

- Handin meets all primary engine requirements
- VoxelSystem
  - Automated chunk streaming
    * Chunks within a certain distance of a selected focus point are automatically loaded (created)
    * Loaded chunks outside a certain distance are unloaded (deleted) from the World

Secondary Game Requirements

- Handin meets all primary game requirements
- Game must incorporate chunk streaming
- Simplistic enemies
  - Spawn either at the beginning of the game or at intervals during play
  - Collide and interact with both the world and the player
  - No complex AI required, but cannot remain stationary or jump in place
  - It must be possible to restart the game without quitting the program
- The game must have at least two screens
Week 4 - Due March. 6

For Minecraft4 you add orthographic UI, heads up displays, and out of game UI to your game. You will also implement additional game features of your choice to add some spice to your voxel world.

Design Check

• How will you integrate orthographic UI, heads up displays, and out of game UI into Minecraft4?
• What types of orthographic UI and heads up displays do you plan to use in your game?
• What extra game features do you plan on implementing

Playtesting Requirements

• Handin only crashes on edge cases
• All Minecraft3 primary requirements must be met

Primary Engine Requirements

• Handin meets all global requirements
• UI
  – Support for orthographic UI within your engine
  – Support for HUDs within your engine
  – Support for basic out of game UI within your engine. Might include implementations for basic UI elements such as
    * Buttons
    * Text boxes
    * Containers

Primary Game Requirements

• Playtesting requirements must be fulfilled
• Your game makes use of all of
  – Orthographic UI
  – HUDs
  – Out of game UI
Secondary Requirements

- Handin meets all primary engine requirements
- Choose one or more of
  - Sound
  - Chunk persistence
  - At least 3 different types of enemies that behave distinctly from one another
  - More interesting procedural terrain. Examples include
    * Using more complex noise algorithm (e.g. 3D noise).
    * Making use of value noise to generate more interesting features in your terrain (e.g. biomes).
  - Something else (run it by the TAs first!)

Handing In

Hand in the entire directory tree for your project, including both your engine and game code. You must also include a README file that describes how to verify each requirement, and an INSTRUCTIONS file that describes how to play your game, as specified in the Global Requirements. To hand in, run `cs195u_handin minecraft n` from the top level directory of your project (which should be where your Qt pro file is), where `n` is the checkpoint you are handing in. Please do not hand in the build files from your project. The game must have at least two screens.