RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

Paper Minecraft

Submitted by:
Ahamed N (201501004).
Vishwamalyan J S (201501059).
Yeswanth C (201501061).
Sunil Kumar S (201501053).

AI19P52 - AI for Game Programming

Department of Artificial Intelligence and Machine Learning

Rajalakshmi Engineering College, Thandalam.

BONAFIDE CERTIFICATE

This is to certify that the Mini project work titled "**Paper Minecraft**" done by Ahamed N – 201501004, Vishwamalyan J S – 201501059, Sunil Kumar S – 201501053, Yeswanth C – 201501061 **(AIML)**, is a record of bonafide work carried out by him/her under my supervision as a part of MINI PROJECT for the subject titled AI19P52/AI for Game Programming by Department of Artificial Intelligence and Machine Learning.

Dr. Bhagavathi Priya S
HEAD OF THE DEPARTMENT
Artificial Intelligence and Machine Learning,
Rajalakshmi Engineering College,
College, Thandalam,
Chennai – 602 105.

Mrs. Tupili Sangeetha
FACULTY IN CHARGE
Assistant Professor
Artificial Intelligence and Machine Learning,
Rajalakshmi Engineering
Thandalam,
Chennai – 602 105.

This project report is submitted for practical examination for AI19P52/ AI for Game Programming to be held on…….at Rajalakshmi Engineering College, Thandalam.

EXTERNAL EXAMINER **INTERNAL EXAMINER**

TABLE OF CONTENTS

ABSTRACT

In Everyday life we come across many hardships and facing many difficulties. As a stress relief to all these hurdles, we introduce a lightweight Minecraft game ‘paper Minecraft’. For children particularly, it could be said that the activities in paper Minecraft can make them smarter and even get them ahead at school. Although most games will promote problem-solving skills to an extent, Minecraft is unique in that it allows players to set their own goals and offers an enormous amount of freedom in how they meet challenges. This game helps in reinforcing problem-solving skills. when it comes to building a shelter, there are countless options, depending on the player’s goal and play style. They could carve out a small burrow or construct a simple dirt hut and wait it out until daybreak. They might continue digging down and spend the night mining. Alternatively, they could commit to building a home, which could range from a simple wooden hut to an elaborate fortress. In normal Minecraft, players will quickly need to learn resource management. Finding and gathering the right resources for specific projects takes time, with the most valuable blocks being the rarest to find. Diamonds, for example, take a specific mining tool and often require players to dig as far down as the level will allow them to go. Even then, it can take half an hour or more to locate a diamond vein, which will typically only provide eight blocks. But as in our simple paper Minecraft, it is unlimited of resources which will help you build and construct units as much as you want.

CHAPTER 1

INTRODUCTION

Minecraft is a fun game that allows for limitless possibilities and creativity. The problem is when children get addicted. Since Minecraft is an open-world sandbox game where almost anything is possible, children get a rush when they can experiment with things they couldn't do in the real world, like building massive Buildings. Minecraft creates an immersive world that allows you to escape from your real-life challenges and problems, at least for a little while. This reduces stress and helps us to relax. Even though the game comes with challenges and stressors of its own, a healthy subconscious mind can differentiate between the fact of the natural world around us and the fictitious threats of the game.

University College London found that games like Minecraft i.e., Our Paper Minecraft game, improves spatial awareness in the brain. The study even found that London taxi drivers that played games like Minecraft in their spare time remembered roads more quickly and avoided accidents more commonly than those who didn't.

This improved spatial awareness increases a child's ability to learn from their environment. Still, it's also beneficial to adults (if you frequently forget where you left things like your keys, you could play Minecraft to improve your spatial awareness). The famous psychologist, Dr. Brian Sutton-Smith, made the statement that the opposite of "play" isn't "work" but "depression." He was a well-known specialist in the areas of play therapy for both children and adults, and he found that playing games (including Minecraft) could counteract the factors that lead to clinical depression.

An unlimited supply of tools and building materials are available in creative mode. You can fly around the biomes exploring various terrains like woods, water, rock, and fields. This game mode is perfect if all you want to do is build cool stuff and share your creations without having to worry about surviving.

CHAPTER 2

LITERATURE SURVEY

Minecraft is a game where the player uses their imagination to shape the world around them using the resources they find. In the world everything is shaped like a block or a cube, this includes the trees, the dirt, the water, the clouds, even the sun, everything. Like Original Minecraft, Paper Minecraft is a open- world dungeon game with lots of features and lightweight which can run any low end PC's. Despite its simple design, Minecraft can be used for a variety of educational purposes. For example, the game can help players learn about architecture and engineering. By building structures in the game, players can experiment with different designs and see how they hold up against the elements. This can teach players about the importance of planning and designing cohesive structures. It can also be used to teach players about geography and geology. The world of Minecraft is made up of biomes, which are specific geographical areas with their own climate and terrain. Players can explore these different biomes and learn about the plants and animals that live there. This can teach players about the diversity of our planet and the interdependence of all living things.

In creating this awesome model, we used pygame which is a library available in python programming language for building games and creating standalone projects. Pygame started in the summer of 2000. Being a C programmer of many years. It will likely even surprise you how much is possible in under 30 milliseconds. Still, it is not hard to reach the ceiling once your game begins to get more complex. Any game running in Realtime will be making full use of the computer. Over the past several years there has been an interesting trend in game development, the move towards higher level languages. Usually, a game is split into two major parts. The game engine, which must be as fast as possible, and the game logic, which makes the engine actually do something. It wasn't long ago when the engine of a game was written in assembly, with portions written in C. Nowadays, C has moved to the game engine, while often the game itself is written in higher level scripting languages. Games like Quake3 and Unreal run these scripts as portable bytecode. Pygame and SDL serve as an excellent C engine for 2D games. Games will still find the largest part of their runtime is spent inside SDL handling the graphics. SDL can take advantage of graphics hardware acceleration. Enabling this can change a game from running around 40 frames per second to over 200 frames per second. When you

see your Python game running at 200 frames per second, you realize that Python and games can work together. It is impressive how well both Python and SDL work on multiple platforms. A chunk is a 384-block tall 16×16 segment of a world. Chunks are the method used by the world generator to divide maps into manageable pieces.

Chunks are 16 blocks wide, 16 blocks long, 384 blocks high, and 98,304 blocks total. They extend from the bottom of the world, Y=-64, all the way up to the build limit of Y=320. Chunks generate around players when they first enter the world. As they wander around the world, new chunks generate as needed. Chunks generate with the help of the map seed, which means that the chunks are always the same if you would use the same seed again, as long as the map generator and version number remain the same. Spawn chunks, a 19x19 set of chunks around the world spawn, are always loaded, so you can use that to your advantage when making automatic farms. the game loads only certain chunks in order to make the game playable. Unloaded chunks are unprocessed by the game and do not process any of the game aspects. For example, in a single-player game with a render distance of 5 chunks, an area of 7×7 chunks centered around the player has a load type of entity ticking (level 31) the immediately enclosing chunks. The strip of chunks at the outer edge of a 9×9 perimeter surrounding the player have ticking (level 32), and the next enclosing chunks (11×11 perimeter) are border chunks (level 33).

In a normal chunk-based terrain, the player moves around in the chunks and chunks are loaded and unloaded depending on some algorithm/methodology. In this alternate method, there are a fixed number of chunks in existence. For the sake of simplicity, let's pretend we have a chunk-view-distance of 1, i.e., 8 chunks around the player and the one he's standing on for a total of 9. Instead of allowing the player to roam around infinite chunks, the chunks are replaced with others as the player proceeds in a direction. This could mean the player never moves, and some movement buffer dictates chunk changing, or it triggers when the player moves over a chunk boundary and then resets the player to the center (like many FPS mouse schemes, but with a player in 3D).

CHAPTER 3

MODEL ARCHITECTURE

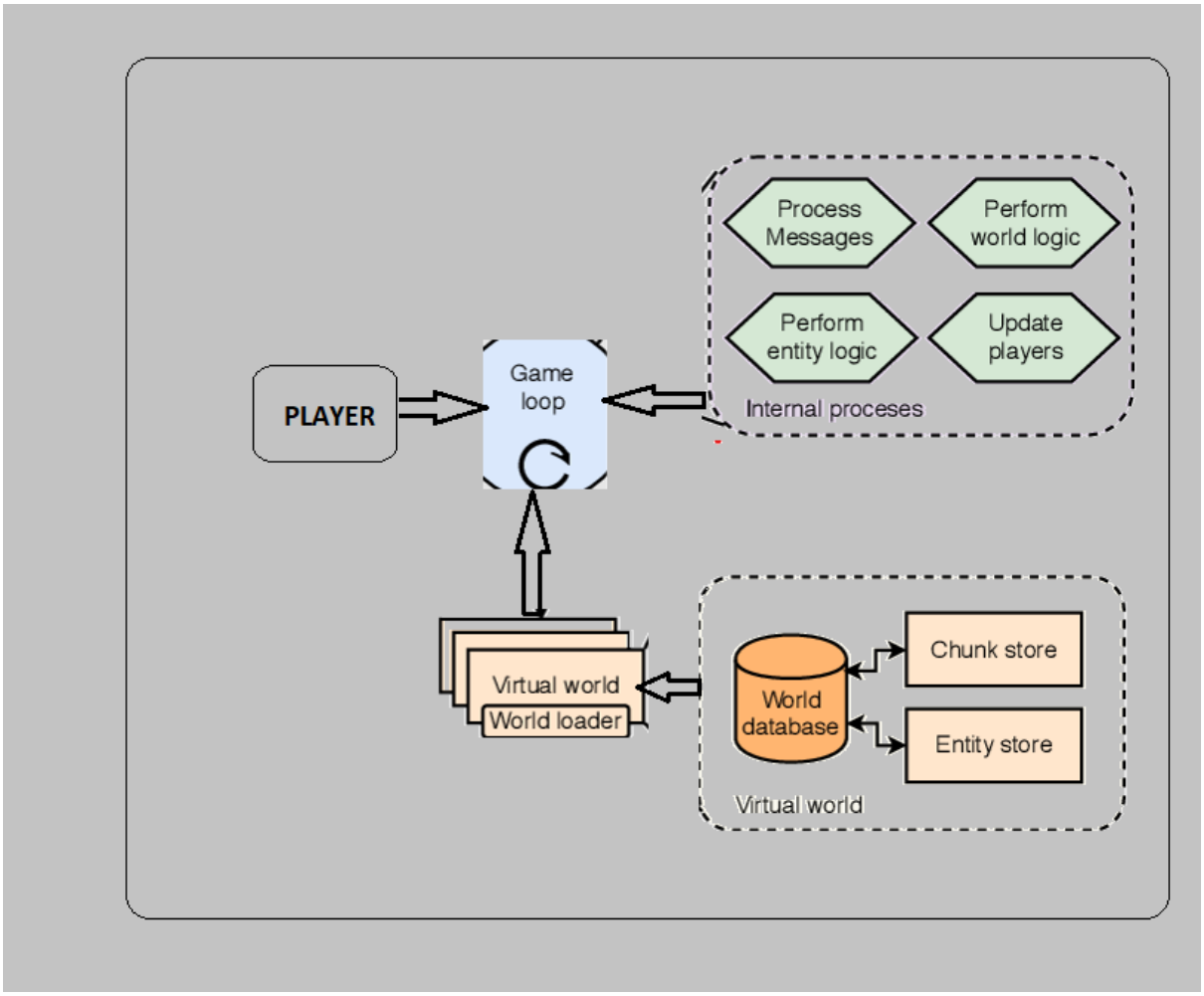
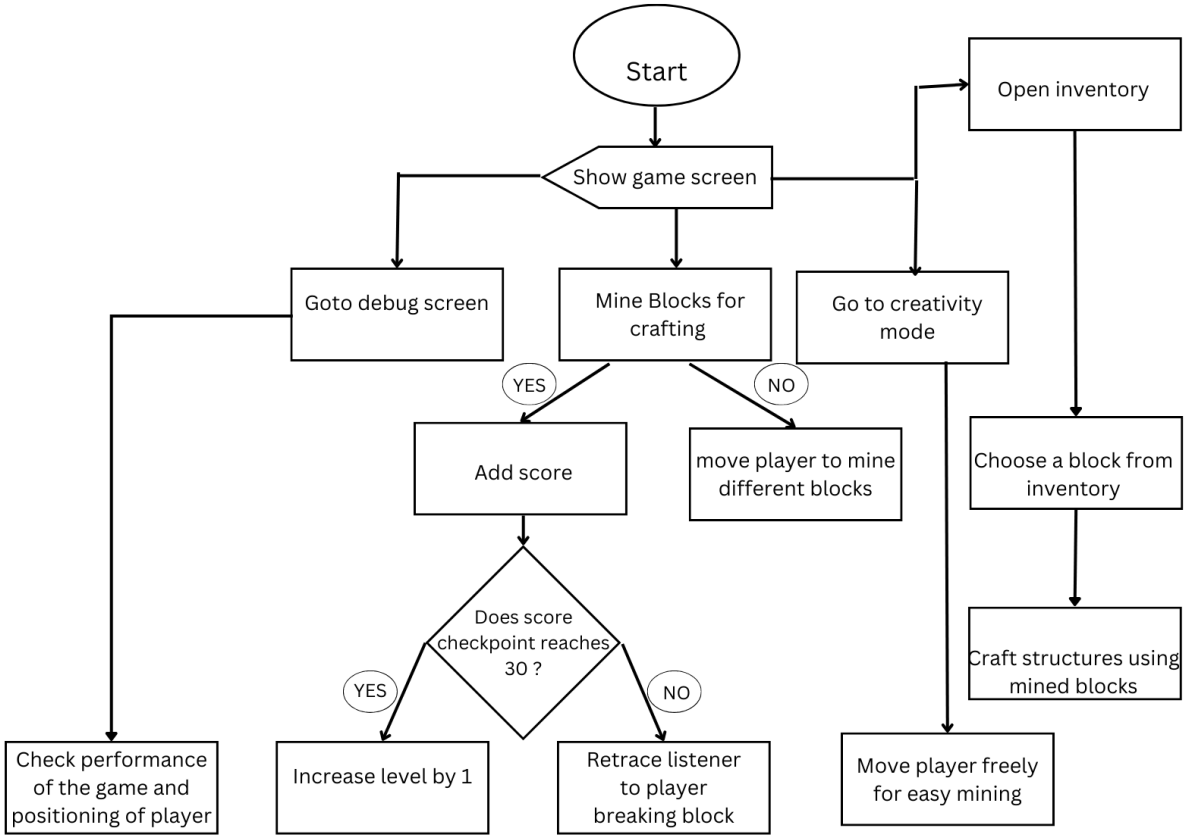


Fig 1: Basic Architecture needed to generate the game world and render player along with the blocks as chunks and entities.

Fig. 1 represents our proposed system architecture where a game loop will be created using pygame and the player will be added as a entity to the environment which is to be rendered with the world generation. The world loader module will take care of loading the chunk details and also added to the game loop. The internal process of this world includes updating the player, score along environment. It also includes applying the game logic into the world.

USE CASE DIAGRAM:



CHAPTER 4
IMPLEMENTATION

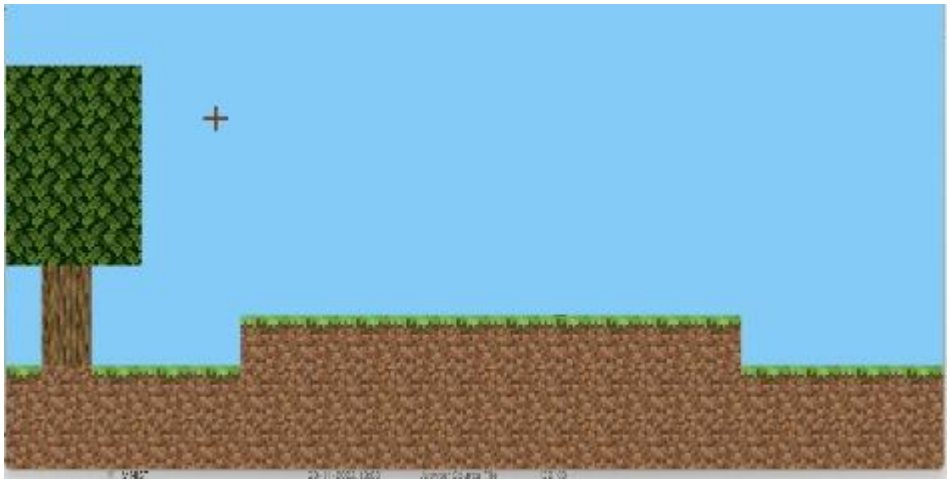
Stage 1 – World Generator:

The majority of the game assets are properties of Mojang Studios. To load the world, we need the objects that are to be present in the world. The chunks and entities needed are block, background, images inventory, particles, players, sprite, utilities etc. after importing them, we create an instance of a main class and proceed to initiate all the values to be loaded. Here chunk load comes into play and add the chunks one by one into the instance with the help of the class. Here by chunks we mean the blocks and these blocks will be distributed by the max_chunks function we have defined.

It Generates chunk data that includes a structure at the given origin. The arguments are origin (tuple): The position of the block where the structure is going to start generating from chunk_pos (tuple): The original chunk position of the structure. chunk_data (dict): The block data of the chunk the structure is originally in. It returns Structure | None: A Structure object with the resulting block data.

Then the generate function, which has Args of block_pos (tuple): the position of the block to generate block_name (str): the name of the block to generate chunk_pos. (tuple): the position of the chunk that the structure originated from chunk_data. (dict): the block data of the chunk that the structure originated from. It returns int: (1 means valid generation, 2 means invalid and the entire structure should not generate, 3 means invalid and only this specific block should not generate, 4 means valid but it would change to another block to replace the target block)

After loading chunks, we have to load camera in the world by using Arg of camera (Camera): The camera used to calculate which chunks should be rendered and returns list: The list of rendered chunks.



Code to generate the structure:

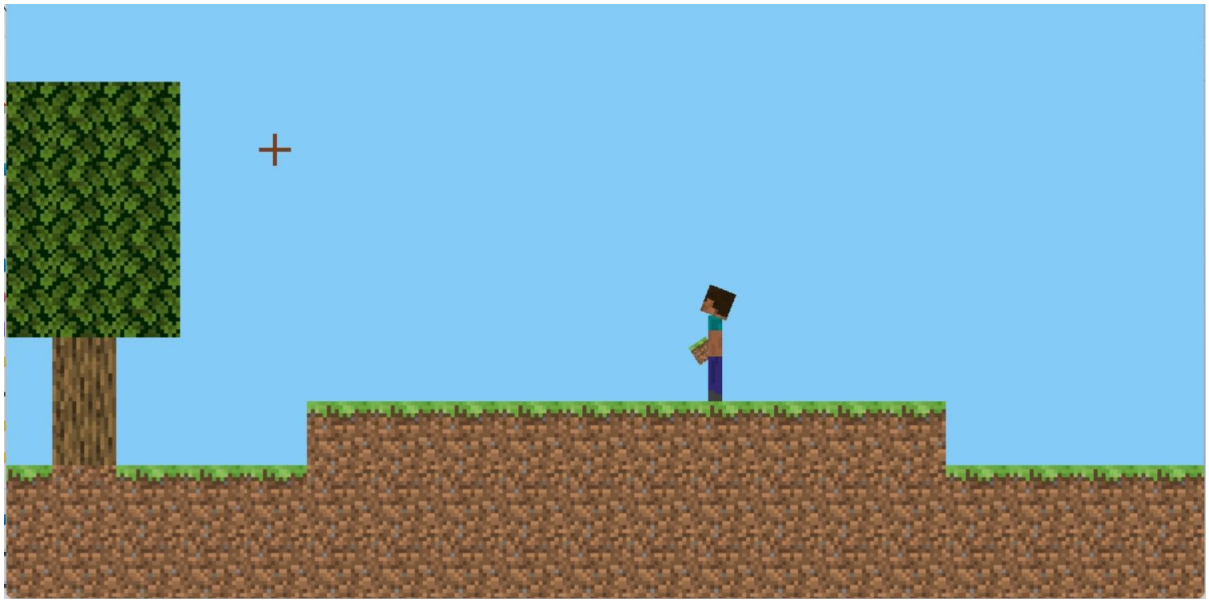
```
def generate_structures(x: int, y: int, chunk_data: dict, name: str, attempts:
int, chance: int | None = None, dist: dict = {}) -> dict:

    generator: StructureGenerator = structure_generators[name]
    if (x, y) not in Structure.instances:
        structs = get_structures(x, y, generator, attempts, chance, dist)

    else:
        structs = [structure.block_data for structure in
Structure.instances[(x, y)]]

    for struct in structs:

        for block_pos, block_name in struct.items():

            block_chunk = (floor(block_pos[0] / CHUNK_SIZE),
floor(block_pos[1] / CHUNK_SIZE))

            if block_chunk[0] == x and block_chunk[1] == y:
                chunk_data[block_pos] = block_name

            else:
                if block_chunk in Chunk.instances:
                    set_block(Chunk.instances, block_pos, block_name)

                else:
                    Chunk.generated_blocks[block_pos] = block_name

    return chunk_data
```

Stage 2 – Update player and world:

Every time key is pressed, the player position will be updated and the velocity is calculated based on the movement of blocks per second. The players height and width is based on the image we are using for the player. We place a pygame rect function in the player, so his movements can be tracked and updated. By default we have set the walking speed to 4.3 BPS (blocks per second). And sprinting speed to 5.6 BPS, jump speed to 7 BPS.

Not only the movement of the player is tracked, but also the block which he holds in the hand also updated based on the selection. Gravity and other physics has been given as default of normal world physics. the terminal velocity comes into play while the gravity pulls the player toward the bedrock. There are various conditions given for the movement of the player

- If the player is moving in the horizontal direction.
- A counter for the rotation of legs.
- If the player is running at full speed.
- Rotate the legs out far (40).
- If the player is running at normal speed.
- Rotate the legs out normally (30).
- If the player is moving very slowly.
- Rotate the legs out a little bit (15).
- Rotate the other leg in the opposite direction.
- Rotate the arms slightly less than the legs.
- Rotate the other arm in the opposite direction.



Code to draw and update player:

```
    def update(self, dt: float) -> None:
        mpos = pygame.mouse.get_pos()
        tick_offset = self.player.pos - self.pos - VEC(SCR_DIM) / 2 +
self.player.size / 2

        if -1 < tick_offset.x < 1:
            tick_offset.x = 0
        if -1 < tick_offset.y < 1:
            tick_offset.y = 0
        if not self.player.inventory.visible:
            x_dist, y_dist = mpos[0] - SCR_DIM[0] / 2, mpos[1] - SCR_DIM[1] /
2
        else:
            x_dist, y_dist = 0, 0
        dist_squared = VEC(sign(x_dist) * x_dist ** 2 * 0.6, sign(y_dist) *
y_dist ** 2 * 1.2)

        self.pos += (tick_offset * 2 + VEC(dist_squared) / 300) * dt

    def draw(self, screen: pygame.Surface, **kwargs) -> None:
        if not constants.MANAGER.cinematic.value["CH"] or
self.master.inventory.visible: return
        self.new_color = self.get_avg_color(screen)


        pygame.draw.rect(screen, self.old_color, (self.mpos[0] - 2,
self.mpos[1] - 16, 4, 32))
        pygame.draw.rect(screen, self.old_color, (self.mpos[0] - 16,
self.mpos[1] - 2, 32, 4))
```

Stage 3 - Objects present in the game other than the world and player:

Inventory:

The inventory is a place where multiple types of blocks are available for the user to craft structures and buildings. In the inventory class we will have a add function, set slot, clear slot and match item functions which are all used when the player makes some move with the inventory.

Add item function is used when the player add a new item he mined from the world. If the mined item already exists in the inventory, then match item function is used, set slot and clear slot are used along with the add item function.

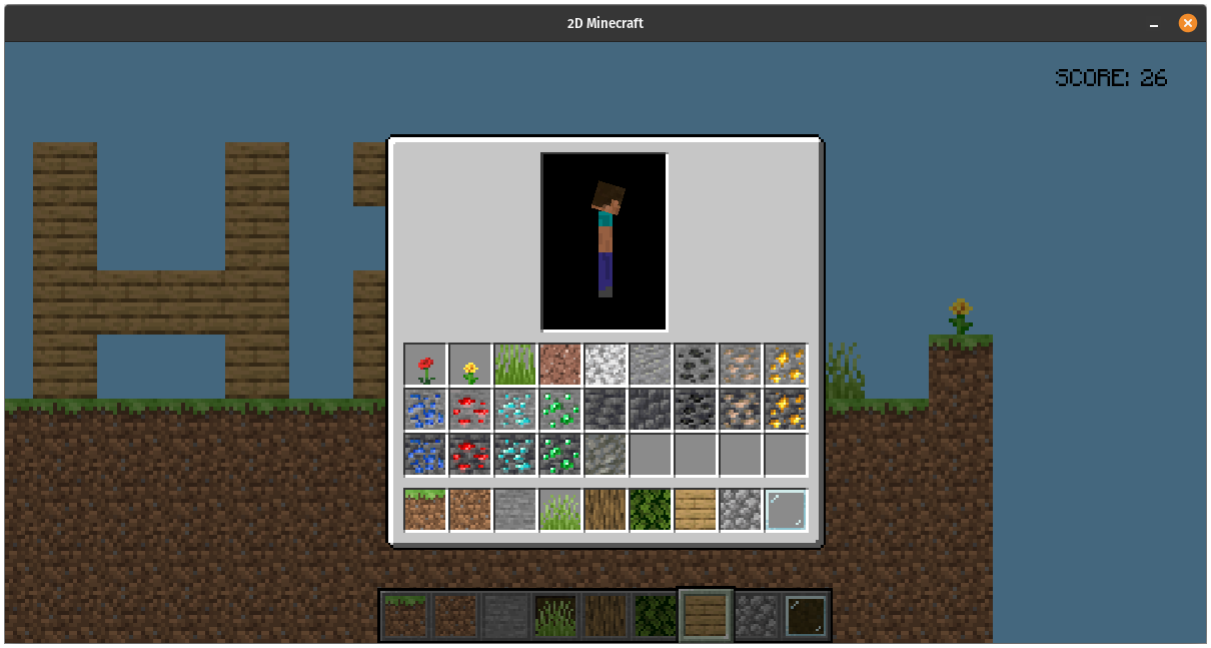


Fig 2: This is the outlook of the inventory which contains multiple block designs

Some of the blocks available are:

- ✓ Tall grass
- ✓ Coal ore
- ✓ Iron ore
- ✓ Gold ore
- ✓ Lapis ore
- ✓ Redstone ore
- ✓ Diamond ore

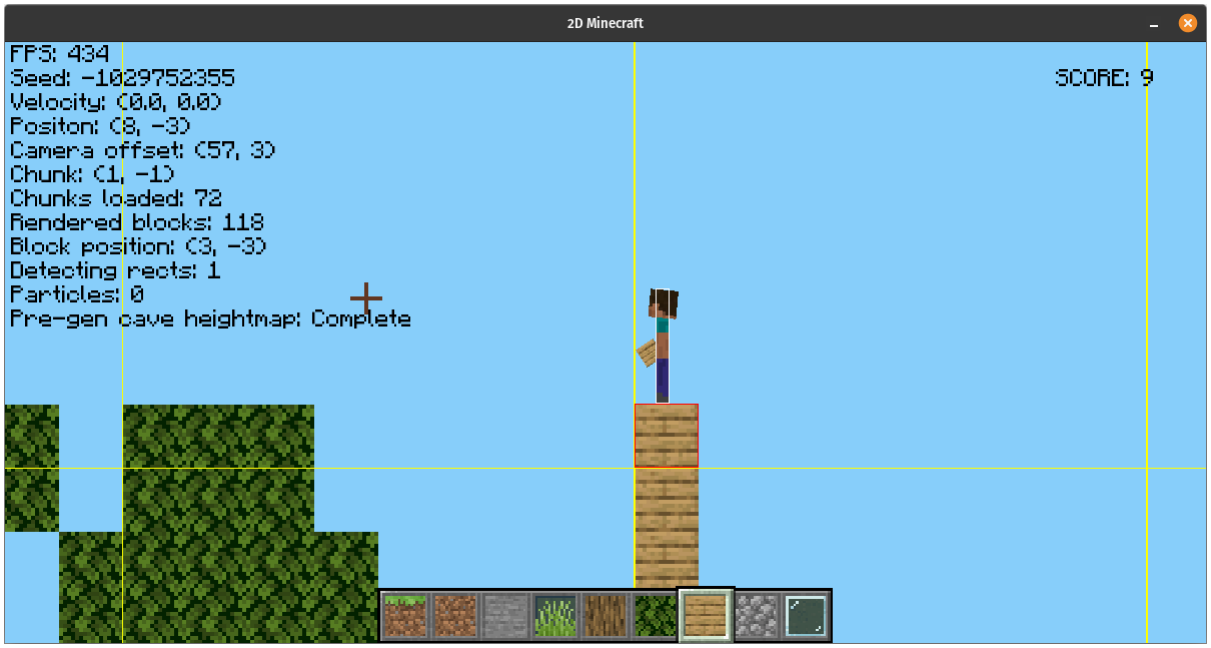
- ✓ Emerald ore
- ✓ Deepslate coal ore
- ✓ Deepslate iron ore
- ✓ Deepslate gold ore
- ✓ Deepslate lapis ore
- ✓ Deepslate Redstone ore
- ✓ Deepslate diamond ore
- ✓ Deepslate emerald ore

Code to generate inventory:

```
class Inventory:
    def __init__(self, max_items: int) -> None:
        self.items = {}
        self.holding = None
        self.max_items = max_items

    def __iadd__(self, other):
        self.add_item(other)
        return self

    def set_slot(self, slot: int, item: str) -> None:
        self.items[slot] = Item(item)

    def clear_slot(self, slot: int) -> None:
        del self.items[slot]

    def add_item(self, item: str | Item) -> None:
        match item:
            case str(item):
                item = Item(item)
            case Item(item):
                pass
            case _:
                raise TypeError("Item must be of type 'str' or 'Item'")

        for y in range(4):
            for x in range(9):
                if (x, y) not in self.items:
                    self.items[(x, y)] = item
                    return

        raise InventoryFullException(item)
```

Debug window:

The debug window is available in our game where we can look at our fps , position, velocity, camera position etc. this greatly helps our players to interact with the game easily and understand what is happening with their window and how their movements affect the gameplay.



Some feature available to look at our debug window are:

- ✓ FPS
- ✓ Seed
- ✓ Velocity
- ✓ Position
- ✓ Camera offset
- ✓ Chunk
- ✓ Rendered block
- ✓ Particles
- ✓ Detecting rect
- ✓ Block position

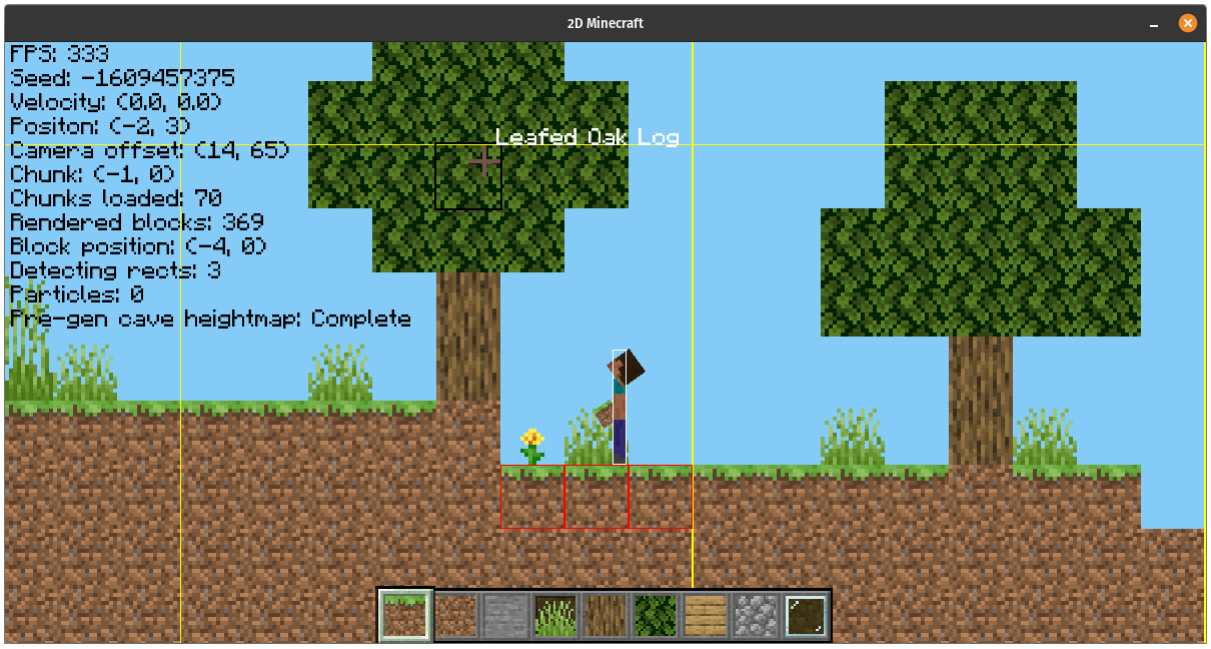
CHAPTER 5
RESULTS AND DISCUSSIONS

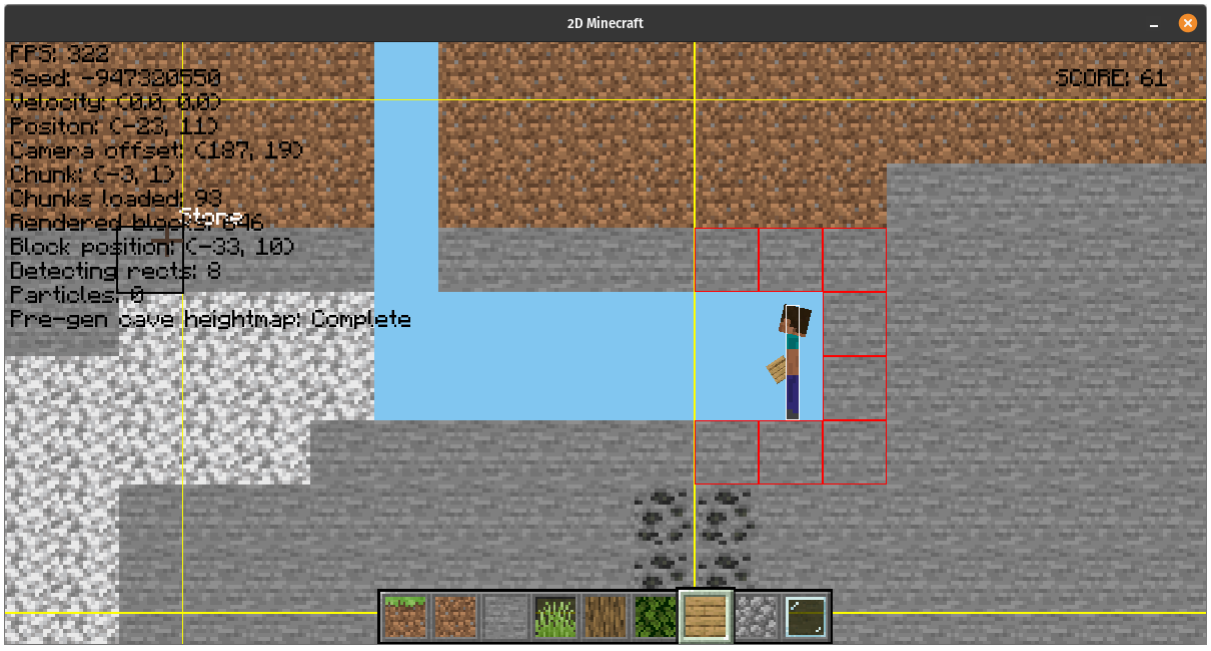
The proposed system structure for the game gives a good performance and also the rendering speed is a prominent feature available in the game as a final result.it has optimized particles with conditional collision testing. Text boxes show their outer border / rect whilst in debug mode. Velocity shows up as BPS (blocks per second) in debug. Optimized chunk loading and structure generation. Spacebar is also bind to jumping. Bedrock is generated with y-1024 height and the world has been made with a definite height.

Terrain Generation has been made optimized. Chunks are generated using Perlin noise as the player moves throughout the world. All block types are used in generation. Game FPS, chunks and blocks loaded are displayed on window title.

When an entity (player) enters a world, it will load a 5x5 area of chunks having the chunk containing the portal in the middle. The outer chunks are lazy chunks. Using this mechanic, it is possible to create machines that send items one way and the other to permanently load the chunks. This type of loading can be improved with more refined class and its instances. In this project we recreated the "Minecraft" game made with scratch in python, while polishing the game, improving the performance and staying true to its 3D counterpart.

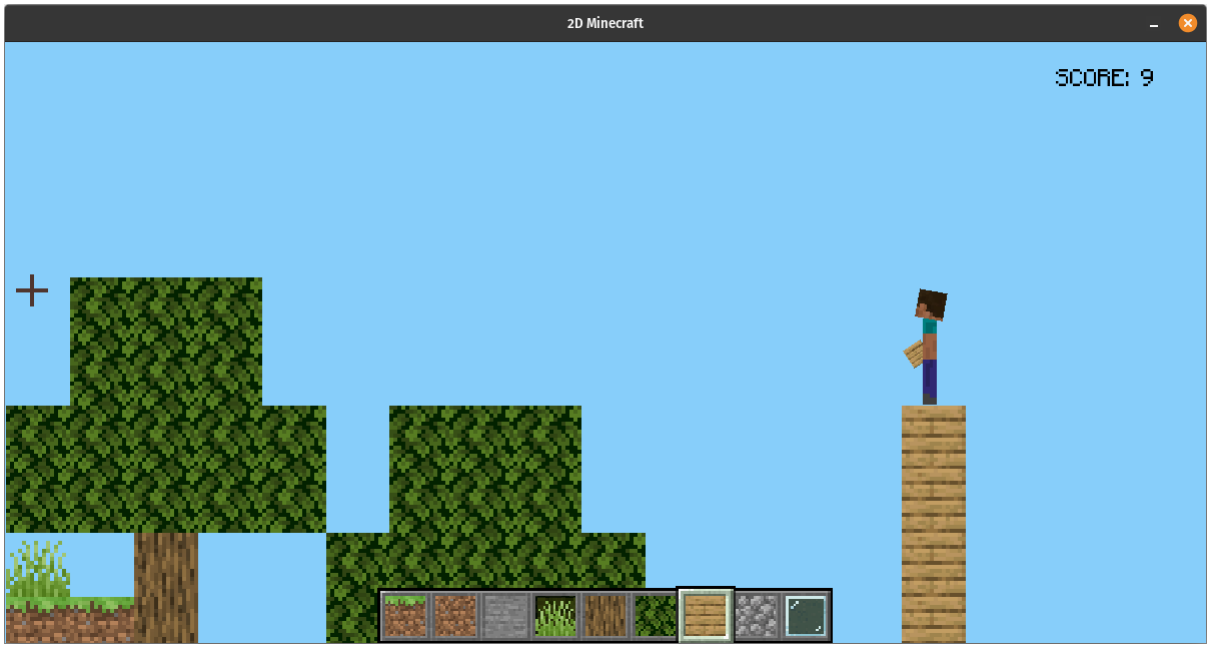
OUTPUT SCREENSHOTS:

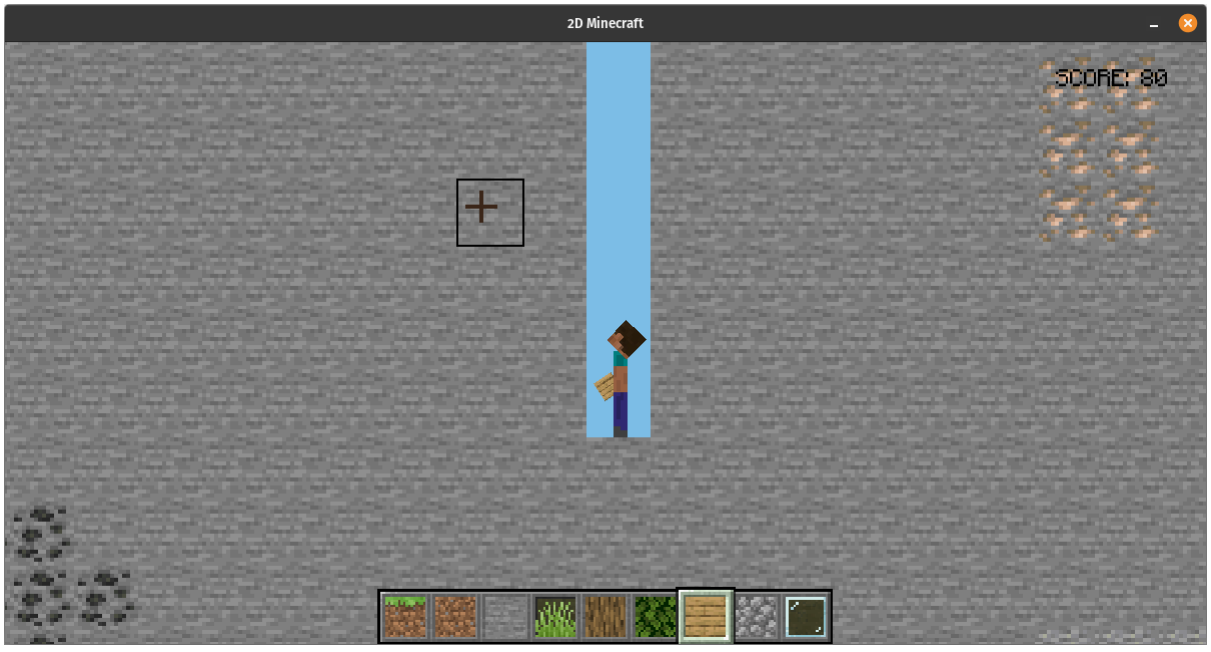





2D Minecraft
Neither


2D Minecraft
SCORE: 40


2D Minecraft
SCORE: 9


2D Minecraft
SCORE: 80

CHAPTER 6
CONCLUSION

The current model of this paper Minecraft project has a lot of features and also implements infinite world building. Even with all the good aspects, we can still improve the world by adding a load feature where once we build a structure, we can store it in a external memory and load it whenever we login into the game. Whatever path the game development moves on, this game is still one monumental project which helps in stress relief and a fun to play open world game.

Real Minecraft doesn't have a score system like this game where in 3D Minecraft the scoring system is completely based our surviving skills and puts some pressure on us. But in this paper Minecraft, the scoring system is completely based on your mining and building skills and the level increase provides some real boost to your gameplay and relax our mind. Also this game provides a debug window where the velocity and position of the player continuously updates and also the camera position is visible to us. As a conclusion the game still has place for improvements, but this game also provides nice user experience with a good FPS rate which is completely adapted by our system's FPS. This feature is provided by a pygame module and helps in make the game smooth and have less latency.

REFERENCES

- https://www.pygame.org/docs/ref/surface.html
- https://pythonprogramming.net/pygame-python-3-part-1-intro
- https://www.pygame.org/docs/ref/pygame.html
- https://github.com/DoubleFaceProgramming/tP7AhUZwTgGHYaMB
- https://minecraft.fandom.com/wiki/List_of_block_textures#:~:text=%E2%80%8C%20%5BBE%20only%5D,Animated%20Textures,Lava%20Fire%20Soul%20Fire%20Kelp
- https://minecraft.fandom.com/wiki/Chunk#:~:text=Chunk%20loading,-Unloaded%20chunks%20in&text=Since%20Minecraft%20worlds%20are%2030,any%20of%20the%20game%20aspects.
- https://www.minecraftforum.net/forums/minecraft-java-edition/discussion/185435-how-many-chunks-around-0-0-are-always-loaded
- https://www.researchgate.net/publication/282778485_Minecraft_computer_game_simulation_and_network_performance_analysis
- https://www.worldql.com/posts/2021-08-worldql-scalable-minecraft/
- https://account.mojang.com/documents/minecraft_eula