AlgoProg Final Project - Imanuel Sheva G Simanjuntak - 2802499592

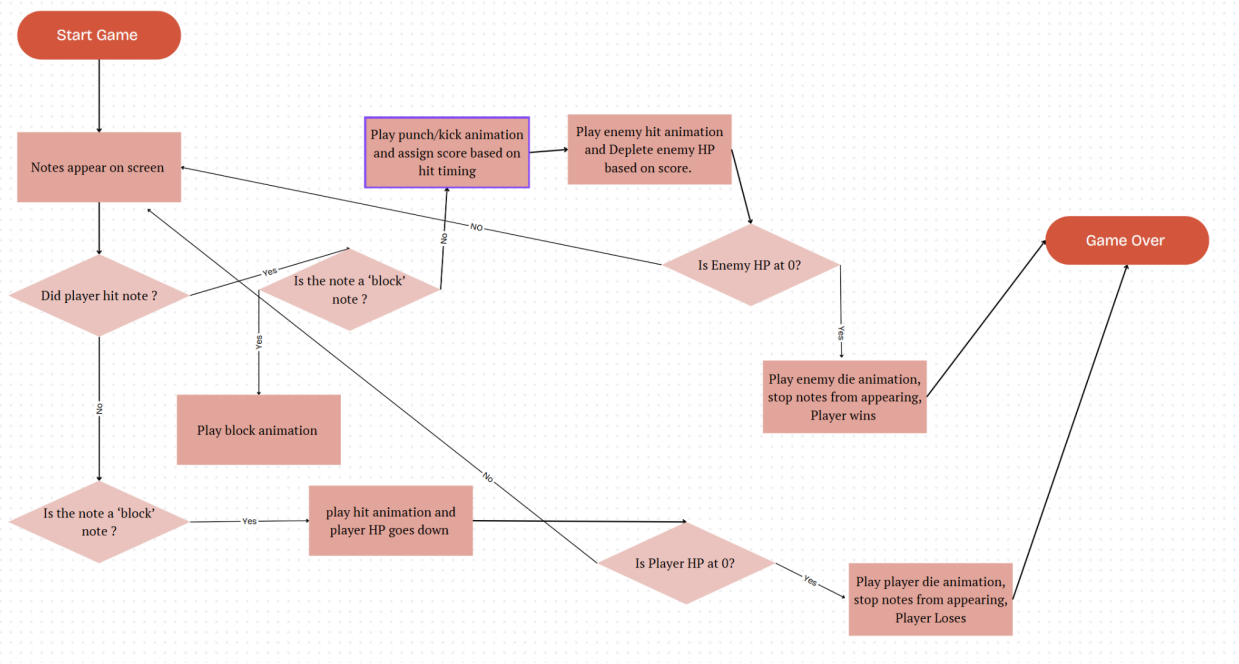
Project Specification

This project is a rhythm-based game where the player engages in a timed sequence of interactions by pressing specific keys to perform actions such as punches, kicks, and blocks. The enemy character reacts dynamically based on the player's input, using predefined animations and behaviors. The primary objective is to defeat the enemy by depleting its health bar while maintaining the player's own health within a set time limit. If the timer runs out and the enemy still has not died, the player loses.

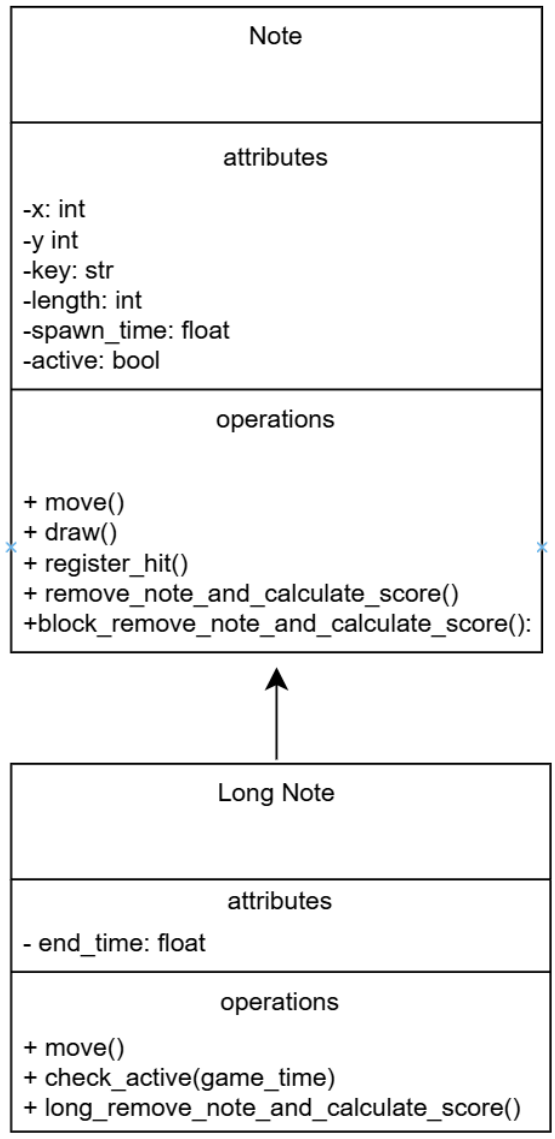
Key Features:

- Interactive Gameplay: Players can punch, kick, and block in response to notes that move across the screen.
- Dynamic Enemy Reactions: The enemy character performs actions such as blocking, hitting, or kicking based on probabilities and player inputs.
- Health Bars: Visual indicators for both player and enemy health, which update in real-time based on gameplay.
- Timer: A countdown timer to ensure each game session is limited.
- Score Evaluation: The game evaluates player performance based on accuracy (Perfect, Great, Good, Miss).

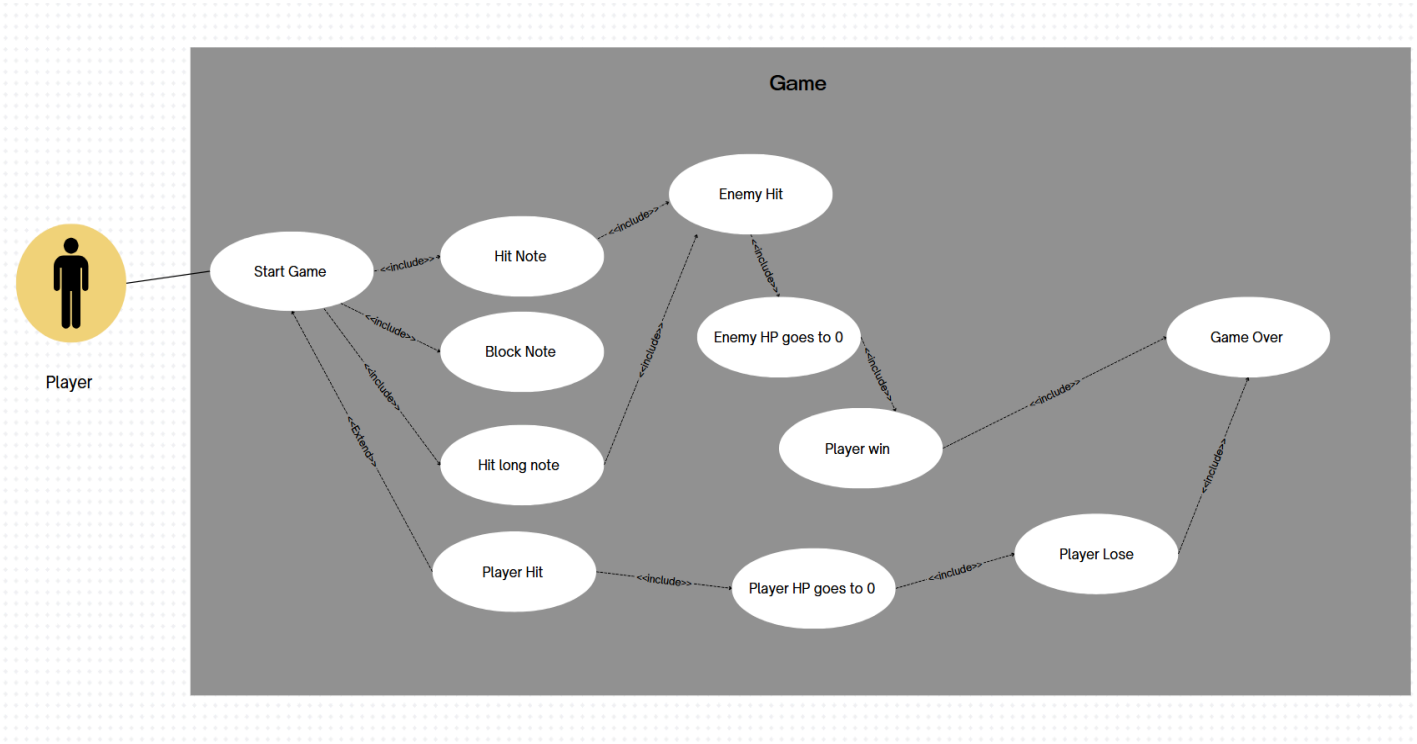
Activity Diagram



UML Class Diagram



Use-Case diagram



Core Algorithms and Logic:

1. Animation System:
   Each character animation is represented by a series of frames. Timers control the frame updates to create smooth transitions.

   Example:

```
if current_frame < len(animation_frames):
    screen.blit(animation_frames[current_frame], position)
    animation_timer += delta_time
    if animation_timer >= animation_speed:
        current_frame += 1
        animation_timer = 0
else:
    current_frame = 0
```

2. Note Interaction:
   Notes are generated based on a predefined pattern and move across the screen. When a note reaches a certain threshold, the player must press the corresponding key to interact with it. A function is then called to calculate if the player hit the note in time and gives it a score (perfect, great, good, miss)

   Example:

```
for note in notes:
    if note.x < GOOD_HIT_THRESHOLD:
        if player_key_press == note.key:
            note.remove()
            calculate_score(note)
```

```python
def remove_note_and_calculate_score(self, key):
    # Remove the note and calculate the score based on key match and position
    global  perfect, great, good, miss, enemy_hit, player_hitted  # Use global variables

    if key == self.key:  # Only check for scoring if the key matches
        if self.x <= PERFECT_HIT_THRESHOLD:
            # Award perfect score if the note is hit at the perfect position
            perfect = True
            removed.append(self)  # Remove the note
        elif self.x <= GREAT_HIT_THRESHOLD:
            # Award great score if the note is hit at the great position
            great = True
            removed.append(self)  # Remove the note
        elif self.x <= GOOD_HIT_THRESHOLD:
            # Award good score if the note is hit at the good position
            good = True
            removed.append(self)  # Remove the note
    else:
        if self.x < 100:  # Check for miss if the key does not match
            # Award miss score if the note is not hit in time
            miss = True
            removed.append(self)  # Remove the note
```

3. Enemy Behavior:
   The enemy reacts based on probabilities influenced by the player's performance. Based on the player's score (by timing the note), the better the score, the higher chance that the enemy will be hit. If enemy_hit is true, its health bar goes down by a bit. If the enemy's health bar goes to 0 the player wins.

```python
if perfect:
    if random.random() <= 0.3:
        e_blocking = True
    else:
        enemy_hit = True
```

```python
if enemy_hit:
    # Increment enemy health if hit
    e_hp_x += 10  # Decrease health when hit
```

```
if e_hp_x >= 1180:
    e_hp_x = 1180
    if not Game_Over:
        game_over_time = time.time() - gst  # Capture the time at the game over moment
        Game_Over = True
        Win = True
```

4. Health and Timer Management:
   The health bars update based on player actions and enemy reactions. If the player misses the 'block' notes, his hp goes down. If the player hp goes down completely, the player loses, and if the player hasn't won after 60 seconds, the player loses.

```
for note in notes:
    if note.key in ['a', 's'] and note.x < 50 and note.length == 0:
        hp_x -= 80
        player_hitted = True
        removed.append(note)
```

```
if hp_x <= 20:
    hp_x = 20
    if not Game_Over:
        game_over_time = time.time() - gst  # Capture the time at the game over moment
        Game_Over = True
        Lose = True
```

Chat (CTRL + I) / Edit (CTRL + L)

```
remaining_time = max(0, start_time - int(game_time))
if remaining_time <= 0:
    Game_Over = True
```

5. Note Spawning
   Notes are defined in a list (spawn time, length, image, etc.) , and are spawned from that list.

```
# spawning notes on the map
note_pattern = [
    Note(spawn_time = 0.23, key='q', speed = 20, image = note_Q, length = 0),
Note(spawn_time = 1.73, key='a', speed = 20, image = note_A, length = 0),
LongNote(spawn_time = 3.48, end_time = 4.79, key='w', speed = 20, image = note_W),
Note(spawn_time = 5.23, key='s', speed = 20, image = note_S, length = 0),
Note(spawn_time = 6.73, key='q', speed = 20, image = note_Q, length = 0),
Note(spawn_time = 8.23, key='a', speed = 20, image = note_A, length = 0),
LongNote(spawn_time = 9.98, end_time = 11.33, key='w', speed = 20, image = note_W),
```

```
# Spawn notes if it's their time
for note in note_pattern:
    if game_time >= note.spawn_time and note not in notes:
        notes.append(note)

for note in removed:
    notes.remove(note)

# Move and draw all notes
for note in notes:
    if isinstance(note, LongNote):  # Check if it's a Long
        note.draw()
        screen.blit(note.image, (note.x, note.y))

    note.move()

    if note not in removed:
        screen.blit(note.image, (note.x, note.y))
```

Modules Used:

1. Pygame:
   The primary library used for building the game. It handles graphics rendering, user input, and sound playback.
2. Time:
   Used for time tracking and syncing game elements (such as notes).
3. Random:
   Adds randomness to elements like enemy blocking or hit probability.

Video of program:
https://drive.google.com/file/d/1nqpTk0arL5Uu-gSIb7igTEK-1bUhS7r_/view?usp=sharing

Screenshots of working program

Reflection:
While working on this project, I became more accustomed to pygame and python. I learned more about classes, inheritance, and functions. The project still could do with some extra features like a home screen or a restart button. I hope in the future I can improve as a programmer.