Smiley Design Document 😊

Overview

Smiley is an arcade-style 2D survival game where the player controls a smiley face avatar navigating a procedurally generated world. The game challenges players to survive against increasingly difficult enemies and interact with the environment to restore health and progress through rounds. Players can move, attack enemies, and collect health potions to prolong survival.

The game features:

- Procedural generation of rooms and hallways.
- Incremental difficulty with increasing enemies per round.
- Health management and healing potions.
- Enemy AI that tracks and chases the player.
- Persistent save/load functionality.

Features

1. **World Generation**
    - Procedurally generated rooms and hallways with walls and floors.
    - Avatar placement at the start of the game.
2. **Player Actions**
    - Movement in four directions (WASD).
    - Attacking nearby enemies.
    - Collecting health potions to restore health.
3. **Enemy AI**
    - Enemies track the player's position and move towards them.
    - Enemies deal damage on collision.
4. **Rounds**
    - Survive a round to progress to the next, with more enemies and challenges.
    - Each round lasts 30 seconds.
5. **Heads-Up Display (HUD)**
    - Displays the remaining time, round number, and player health.
6. **Audio Integration**

    - Background music that loops continuously during gameplay.
7. **Save/Load Functionality**
    - Save the game state, including player health, enemy positions, and round number.
    - Load a previously saved game.

Class Design

1. Main Class

Purpose: The main entry point of the game. Handles the game loop, menus, and user input.

Key Methods:

- run(): Main game loop and menu navigation.
- startNewGame(): Initializes a new world and begins the game.
- gameLoop(): Core gameplay loop, handling player input, enemy movement, and HUD rendering.
- renderFrameWithHUD(): Renders the game frame along with the HUD.
- saveGame(World world)/loadGame(): Save and load game state.

---

2. World Class

Purpose: Represents the game world, including rooms, entities, and the player.

Instance Variables:

- world: 2D array of tiles representing the world.
- avatarX, avatarY: Player's current position.
- entities: List of enemies in the world.
- healthPotions: List of health potion positions.
- playerHealth: Player's health.
- roundNumber: Current round number.

Key Methods:

- generateWorld(): Generates the world layout, including rooms and hallways.
- placeEntities(int count): Spawns enemies in random valid positions.
- moveAvatar(char direction): Moves the player based on input.

- moveEntities(): Moves enemies towards the player.
- checkCollision(): Checks for collisions between the player and enemies.
- startNewRound(): Starts a new round with additional enemies and health potions.
- checkForHealthPotion(): Checks if the player collected a health potion.
- getWorldState()/loadFromString(): Serialize and deserialize the game state.

---

3. Entity Class

Purpose: Represents enemies in the game.

Instance Variables:

- x, y: Current position of the enemy.
- tile: Tile representation of the enemy.
- health: Enemy's health.
- damage: Damage dealt by the enemy.

Key Methods:

- takeDamage(int amount): Reduces enemy health.
- isAlive(): Checks if the enemy is still alive.
- moveTowards(int targetX, int targetY, TETile[][] world): Moves the enemy towards the target position.

---

4. AudioPlayer Class

Purpose: Handles background music during the game.

Key Methods:

- play(): Starts playing the audio file.
- stop(): Stops the audio playback.

---

5. Room Class (Inner Class of World)

Purpose: Represents individual rooms in the world.

Instance Variables:

- x, y: Top-left corner of the room.

- width, height: Dimensions of the room.

Key Methods:

- overlaps(Room other): Checks if the room overlaps with another.

---

6. Tile Engine (TETile and Tileset)

Purpose: Provides graphical representation of the game world.

Tiles:

- AVATAR: Represents the player.
- CELL: Represents enemies.
- HEART: Represents health potions.
- FLOOR: Represents walkable areas.
- WALL: Represents walls.

Workflow

1. Main Menu

- **Options**:
    - N: Start a new game.
    - L: Load a saved game.
    - Q: Quit the game.

2. World Generation

- Create a 2D world with procedurally generated rooms and hallways.
- Place the player and initialize entities.

3. Gameplay

- Player can move, attack enemies, and collect health potions.
- Enemies move towards the player every few frames.
- Survive for 30 seconds to advance to the next round.

4. Game Over

- Game ends when the player's health reaches 0 or they are caught by an enemy.

Challenges

1. **AI Pathfinding**: Simplified with direct movement towards the player.
2. **Procedural Generation**: Random room placement while avoiding overlaps.
3. **State Persistence**: Save and load the complete game state, including entities and world layout.

Future Enhancements

1. Add multiple enemy types with different behaviors.
2. Introduce power-ups or weapons for the player.
3. Implement advanced AI for enemies.
4. Improve HUD to display more information (e.g., score or remaining enemies).