SkyPark Game Development Documentation

1. Game Structure

This project is a 3D platformer game, built using **Godot Engine**, lets players navigate floating platforms, collect coins, avoid bombs, and reach checkpoints. There are two variations of the game : Hard mode and Easy mode. The difference between them lies in that in hard mode there are also bombs – hazards that when the player is in contact with them they lose a life. Whereas in easy mode the bombs still persist in the game but the player cannot interact with them.

Scene Hierarchy:

```
Game (Node3D)
├── DirectionalLight3D
├── WorldEnvironment
├── simplePlayer (CharacterBody3D)
├── Coins (Node3D – folder where coins are kept)
├── GridMap
├── checkpoints (Node3D – folder where checkpoints are stored)
├── FallZone (Area3D)
├── HUD (CanvasLayer)
├── movingPlatforms (Node3D – folder where moving platforms are stored)
├── Portals (Node3D)
├── Bombs (Node3D)
└── BackgroundMusic (AudioStreamPlayer3D)
```

- **simplePlayer:** Main character controlled by the player.
- **Coins:** Collectible objects scattered in the world.
- **checkpoints:** Save player progress for respawning.
- **HUD:** Displays the coin count, timer, and health.
- **movingPlatforms:** Platforms that the player can ride.
- **Portals:** Teleport the player between locations.
- **Bombs:** Explode on contact with the player.
- **FallZone:** Detects when the player falls and triggers respawn.

Collision layers and masks

Collision layers and masks are used to define the type of object one object can interact with in the game. This game uses **5 custom collision layers** to manage object interactions efficiently:

1. **Player:** Assigned to the **player** character to detect collisions with the ground, collectibles, checkpoints, and hazards.
2. **Ground:** Includes static and moving platforms, enabling the player to walk and jump.
3. **Collectable:** Used for **coins** and bombs, allowing the player to pick them up or interact with them.
4. **Checkpoint:** Assigned to **checkpoints**, enabling the player to update their respawn point.
5. **FallZone:** Represents the **fall zone** beneath platforms, triggering respawn when the player falls.

2. Main Functionalities & Implementations

2.1 Player Movement

- **Node:** `simplePlayer (CharacterBody3D)`
- **Controls:**
    - **WASD** for movement
    - **Spacebar** for jumping
    - **Mouse** for looking around

Implementation:

- Movement is handled in the `_physics_process` function using `move_and_slide()`.
- Gravity and jumping are managed with velocity adjustments.
- The camera follows the player smoothly, and mouse input rotates the camera and player.

2.2 Checkpoints & Respawn System

- **Node:** `checkpoints (Area3D)`
- **Purpose:** Save player position for respawn.

Implementation:

- When the player enters a checkpoint, a **signal** (`reached`) is emitted.
- The player listens to this signal and saves the current position as the respawn point.
- If the player falls into the **FallZone**, they respawn at the last checkpoint.

2.3 Health System

- **Managed by:** `Global.gd (Singleton)`
- **Purpose:** Track and display player lives.

Implementation:

- The player starts with 3 lives, displayed as hearts in the HUD.
- When the player dies (by falling or hitting a bomb), a function in `Global.gd` reduces lives and emits a **`playerDied`** signal.
- The game listens to this signal to update the HUD and trigger game over when lives reach zero.

2.4 Timer System

- **Node:** `Timer` in `game.gd`
- **Purpose:** Countdown timer that affects gameplay.

Implementation:

- The game starts with a timer (e.g., 225 seconds) displayed in the HUD.

- A **`timeout`** signal from the Timer triggers a function that reduces the player's life when time runs out.
- The timer is displayed and updated every frame.

2.5 Coin Collection

- **Node:** `Coins (Area3D)`
- **Purpose:** Collectibles that contribute to winning the game.

Implementation:

- When the player enters a coin's collision area, the coin increments the **global coin count** and updates the HUD.
- A collection animation plays, and when all coins are collected, the game switches to a **Win** screen.

2.6 Bombs and Explosions

- **Node:** `Bomb (Area3D)`
- **Purpose:** Hazards that damage the player.

Implementation:

- When the player collides with a bomb, the bomb triggers an explosion animation using particles and sound effects.
- After the explosion, a **`respawn`** signal is emitted to respawn the player at the last checkpoint and reduce a life.

2.7 Moving Platforms

- **Node:** `movingPlatforms (StaticBody3D + AnimationPlayer)`
- **Purpose:** Platforms that move and carry the player.

Implementation:

- Platforms are animated using **AnimationPlayer**.
- The player script checks if the player is standing on a moving platform using `get_last_slide_collision()`.

- If so, the platform's velocity is applied to the player to make them move along with it.

2.8 Portals

- **Node:** `Portals (Area3D)`
- **Purpose:** Teleport the player between locations.

Implementation:

- When the player enters a portal, the **`body_entered`** signal is triggered.
- The player's position is set to the linked portal's location with a slight offset to avoid immediate re-entry.

2.9 User Interface (HUD)

- **Node:** `HUD (CanvasLayer)`
- **Purpose:** Display the player's coin count, health, and timer.

Implementation:

- The HUD updates dynamically based on the player's current status.
- Signals from the **Global.gd** singleton (like `playerDied`) and the coin collection system update the HUD in real-time.
- The timer is displayed and counts down in **MM:SS** format.

3. Signals Overview

Godot's signal system is used in this game to allow nodes to communicate with one another without being closely coupled. When the player dies, receives a coin, or reaches a checkpoint, for example, signals assist in initiating actions in response to such events. Here is a quick summary of how I have utilized them in my game:

- **Checkpoint Reached:**
  Each **Checkpoint** node emits a **`reached`** signal when the player enters its area. The **simplePlayer** node listens for this signal to update the player's respawn location. This allows the player to respawn at the last reached checkpoint after dying.

- **Player Death:**
  The **Global.gd** singleton manages the player's health and emits a **`playerDied`** signal whenever the player loses a life, whether due to falling, running out of time, or interacting with hazards like bombs. The **Game** node listens to this signal to update the HUD (health display) and check for game-over conditions.
- **Timer Expiry:**
  A **Timer** node in the **Game** scene counts down the remaining time. When it reaches zero, it emits a **`timeout`** signal, which triggers the function to reduce the player's life, simulating the effect of time running out. This keeps the game challenging and time-bound.
- **Area Collisions (Coins, Bombs, Portals):**
  Various **Area3D** nodes emit **`body_entered`** signals to detect when the player interacts with them:
  - **Coins** use this signal to increase the global coin count and update the HUD.
  - **Bombs** use it to trigger explosions and damage the player.
  - **Portals** use it to teleport the player between different locations.
- **Bomb Explosions:**
  After a bomb explodes, it emits a custom **`respawn`** signal, prompting the **simplePlayer** node to respawn at the last checkpoint. This signal ensures that the player is repositioned only after the explosion finishes, creating a smooth gameplay experience.

4. Game Over Conditions:

As mentioned in the structure summary, the Global.gd script (Singleton design pattern) is used to keep track of the progress in the game. This includes the timer countdown, collection of the hidden treasure or coins, and the lives of the player.
The win condition is reached when the player is able to collect all the coins in the game, while still having at least one remaining life.

Lives are lost when the player falls off the park, or when walking into a bomb. When all the lives are lost, the game is lost as well. A loss screen is displayed to the user.

In cases when the timer runs out, a life is taken from the player, and the new available time given to the user decreases in factors of 2 each time the timer runs out. Example: firstly, the user has 3 minutes and 45 seconds to complete the game, if he is unable to, a life is lost and the new timer is half the size it was before he lost a life in this scenario - 1min 52s.