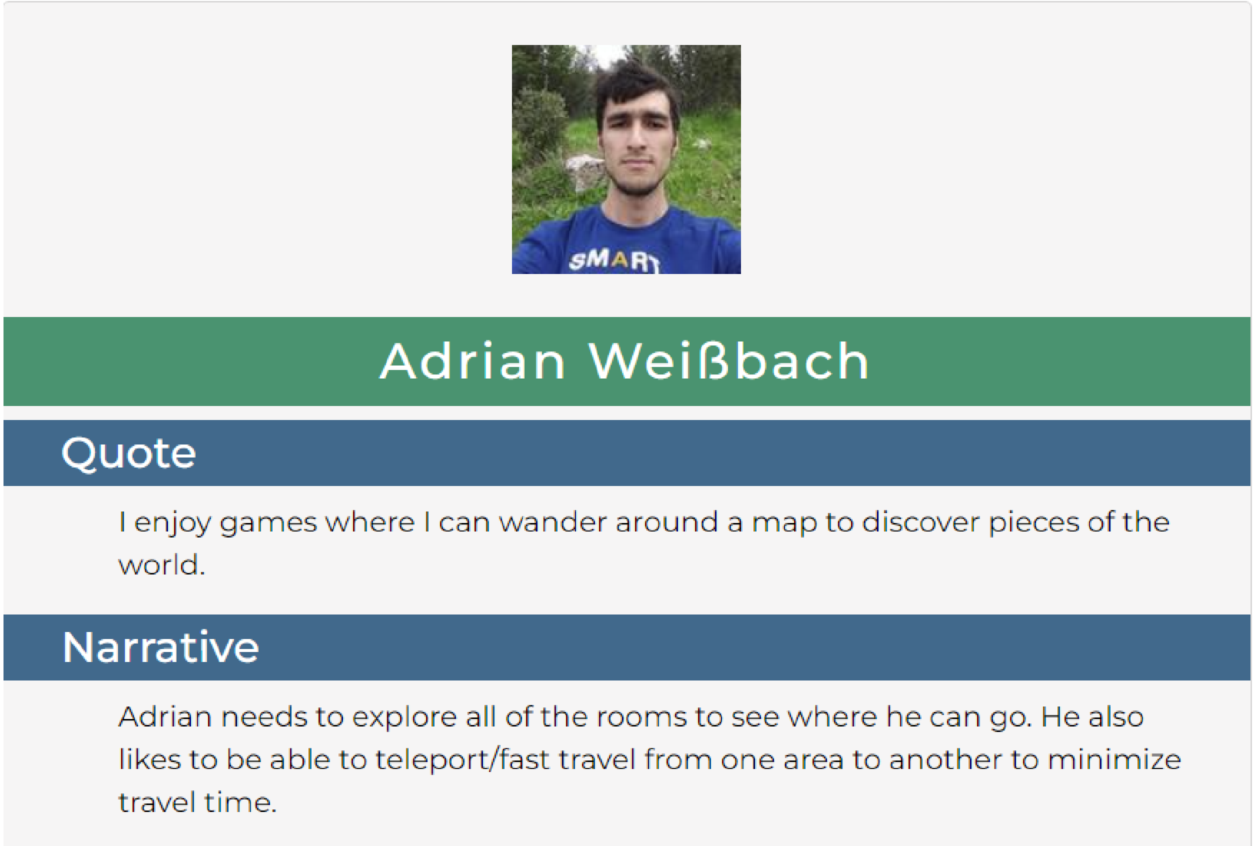
Design Manual: JukeBox Hero

Introduction

JukeBox Hero is a 2D computer game that allows the user to embrace their curiosity and explore a strange maze of rooms as the character TICOO™. They will be able to interact with a number of items and non player characters who populate the halls. Several key characteristics of this game include the beautiful artwork, the complex series of collide-able items that make up the series of rooms, and the puzzles (that have been partially developed and will be available in a future update) that control the locked doors.

User Stories

We mainly focused our work on developing a game for an “Explorer”, a player who wants to learn about the world.



Adrian Weißbach

Quote

I enjoy games where I can wander around a map to discover pieces of the world.

Narrative

Adrian needs to explore all of the rooms to see where he can go. He also likes to be able to teleport/fast travel from one area to another to minimize travel time.

Object Oriented Design

CRC Cards

Player	
Track position in room and move	position, Room
Manage reach boundary	Rectangle

Manage hit box boundary	Rectangle
Manage items in their inventory	Inventory, Items
Contains player-specific texture	Image (sprite)

The player is the user-controlled component of the game, with which the user can interact and explore the world around them. As the player moves, the Image (sprite) is mapped on top of its position and the reach and hit box boundaries are updated as well.

Room	
Tracks all Items in the Room	List, Items
Tracks all components of the Room	List, Door, Wall,

The Room objects store everything the user will ever see in the game (apart from the Player itself), including the NPCs, items, and wall / floor sprites. They allow more spaces to be quickly and efficiency created and viewed throughout the game.

Wall	
Tracks position in Room	Rectangle

The surrounding boundaries of each Room object, to prevent the user from wandering offscreen into the abyss. Automatically surrounds every non-Door section of each Room.

Door	
Tracks room connections	Doorway

Doors are also contained within a Room, and are the vessels that transport the Player from one Room to another. When a Player intersects with a Door’s Doorway objects, it is automatically “teleported” into the next Room, as stored by the Doorways.

Inventory	
Manage items in a set	List of items
Add/remove items	Item types

The Inventory tracks the items held by a given Player, NPC, or object in general. The Player’s Inventory, for example, is updated as the Player picks up and drops items throughout the game.

Puzzle	

Track if its solved	Boolean values
Manage items relevant to puzzle	Inventory, Items

A Puzzle is an overarching base class for the various tasks the Player must complete to progress in the game. While the specific Puzzle subclass may vary significantly, it always is either solved or not and has a set of items pertaining to it.

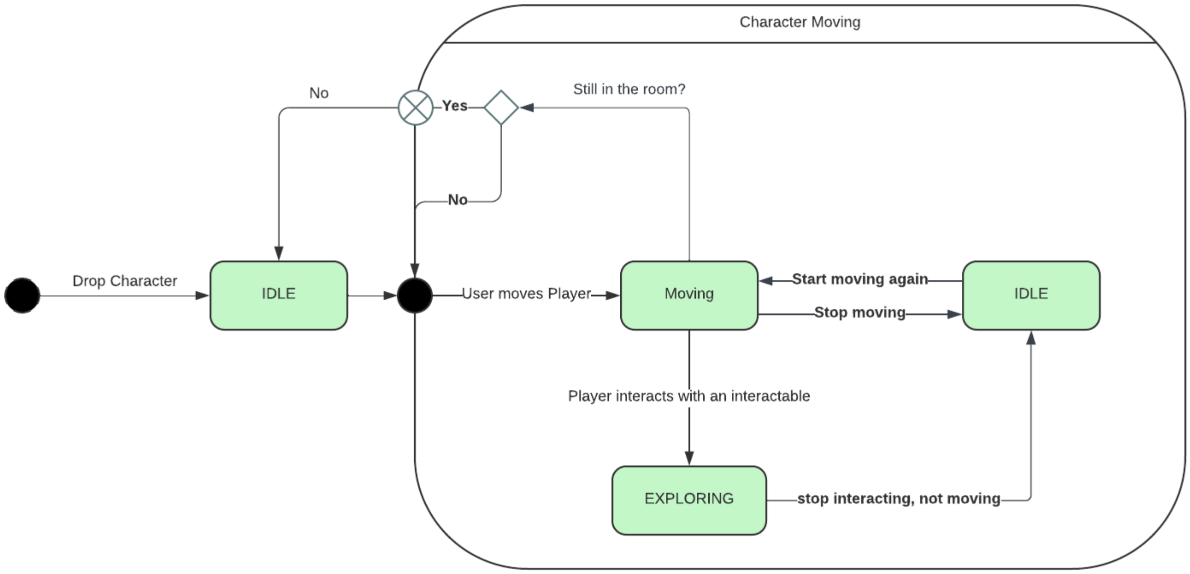
RoomManager	
Generates all Rooms	Room, Wall, Door, Item, etc.
Tracks all Rooms in the game	List, Rooms

A controller class for the Room objects throughout the game that handles constructing, updating, and changing any data pertaining to any of the Rooms in the game as the Player interacts with and in them. This removes a lot of functionality from the Room class itself and allows an individual Room to have no knowledge of any other Rooms explicitly.

Talker	
Interacts with the Player when prompted	Player
Contains text to display	String (messages)

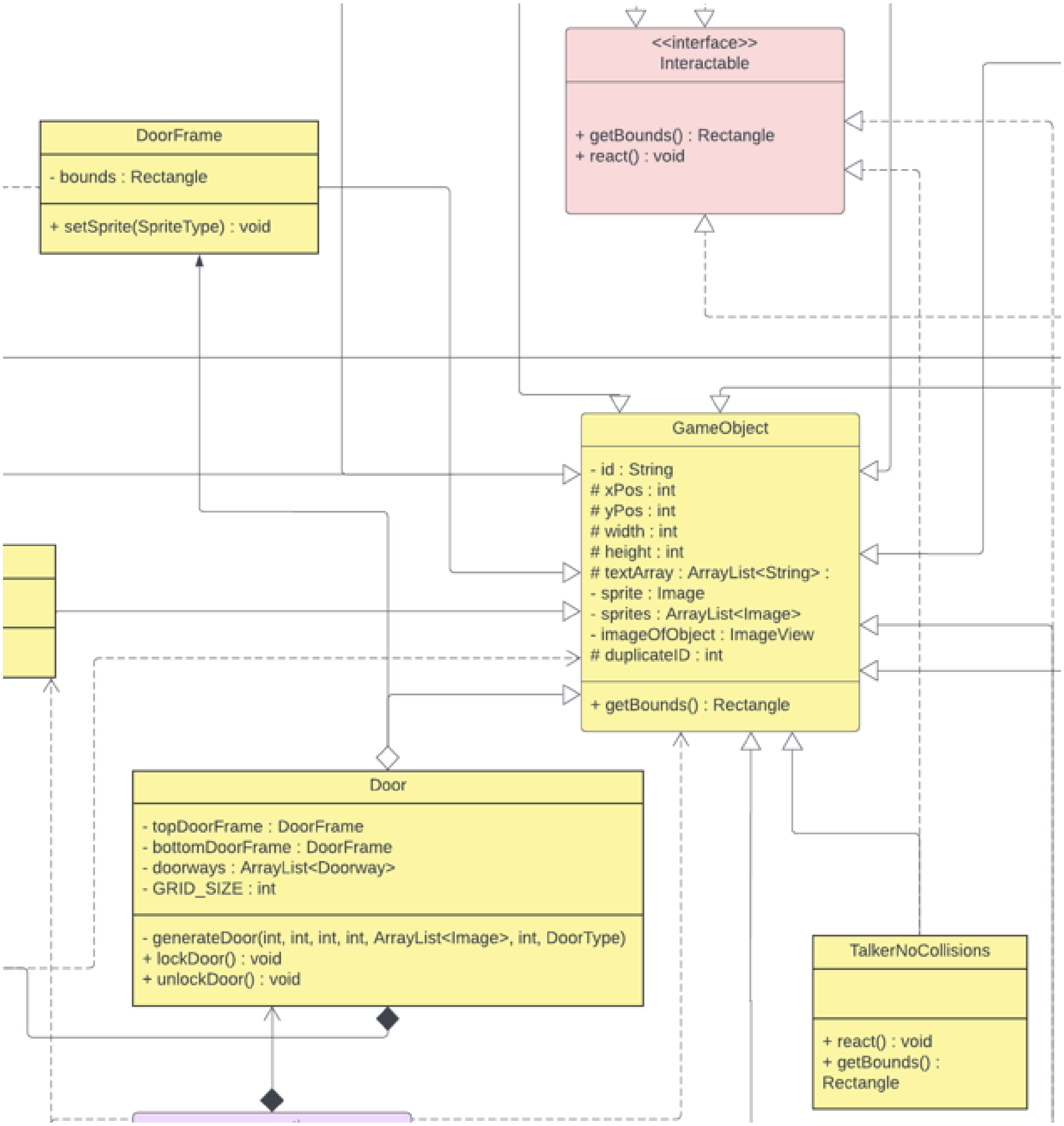
The class for any NPC or item that displays dialogue when the Player wishes to interact with them. This could range from a simple statement of what the object is to a full backstory about who the NPC is and what they are doing.

UML Diagrams : State Diagram



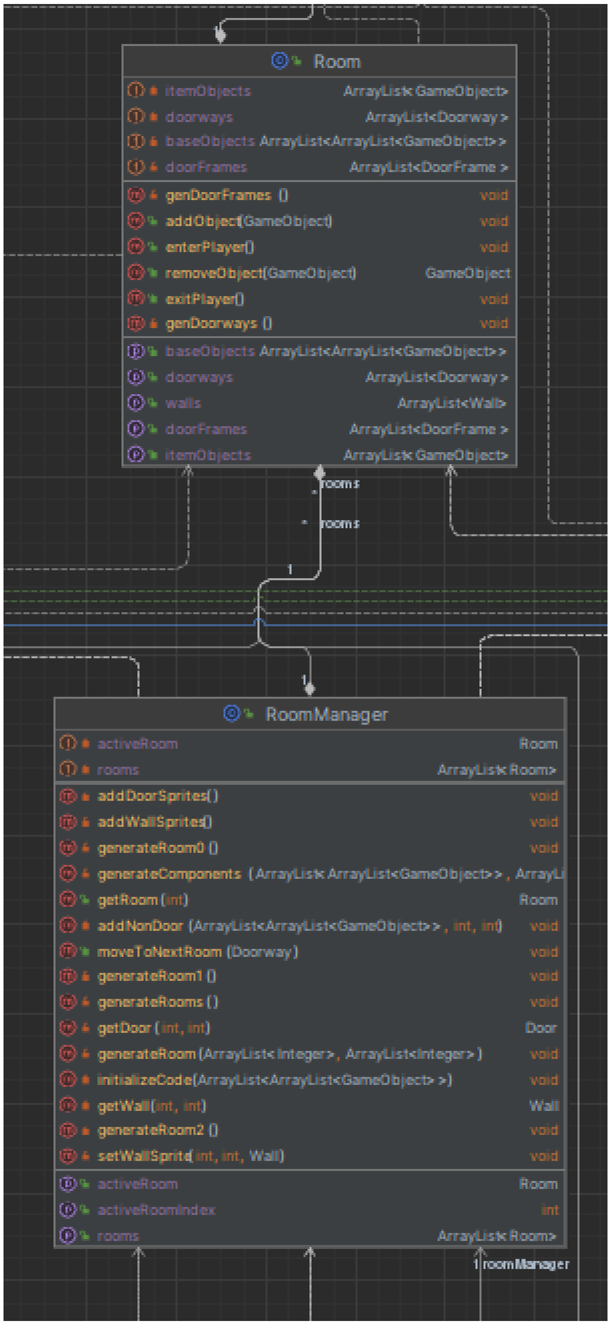
One example of how the program works is pictured in the above state diagram that demonstrates how the player can move the character, TICOO, through the rooms. The character spawns in and is set to IDLE: they are not moving, nor exploring anything in the rooms. As the user begins to move TICOO around the room, they will become in the state of MOVING. There are other states that they may enter into. If TICOO stops moving, their state will be switched to IDLE. If they interact with an object in the room (by picking it up, placing it down, or speaking with it), their state will change to EXPLORING. When TICOO moves out of the room through a door, they will be rendered in the next room as IDLE until they begin moving or exploring that new room.

UML Diagrams : Class Diagram



<<interface>>
Interactable
+ getBounds() : Rectangle
+ react() : void
DoorFrame
- bounds : Rectangle
+ setSprite(SpriteType) : void
GameObject
- id : String
# xPos : int
# yPos : int
# width : int
# height : int
# textArray : ArrayList<String> :
- sprite : Image
- sprites : ArrayList<Image>
- imageOfObject : ImageView
# duplicateID : int
+ getBounds() : Rectangle
Door
- topDoorFrame : DoorFrame
- bottomDoorFrame : DoorFrame
- doorways : ArrayList<Doorway>
- GRID_SIZE : int
- generateDoor(int, int, int, int, ArrayList<Image>, int, DoorType)
+ lockDoor() : void
+ unlockDoor() : void
TalkerNoCollisions
+ react() : void
+ getBounds() : Rectangle

Above is a portion of our meticulously created UML class diagram. Our users mentioned that they liked to explore, so we created a lot of different items and other non player characters in the game to interact with. In order to easily access and manipulate these, we implemented a GameObject class, for all of the other classes to inherit from, so that everything was a bit easier to pass from room to room, or room to player, etc. Also in this UML, we can see one of the interfaces that we implemented - “Interactable”. There are three similar interfaces that most GameObject children implement in some respect: Interactable, Storable, and Collideable. In the diagram, we can see TalkerNoCollisions implements Interactable, meaning that the player can interact in some way, typically by “speaking” with it.



```
Room
itemObjects                     ArrayList<GameObject>
doorways                        ArrayList<Doorway>
baseObjects ArrayList<ArrayList<GameObject>>
doorFrames                      ArrayList<DoorFrame>
genDoorFrames ()                void
addObject(GameObject)           void
enterPlayer()                   void
removeObject(GameObject)        GameObject
exitPlayer()                    void
genDoorways ()                  void
baseObjects ArrayList<ArrayList<GameObject>>
doorways                        ArrayList<Doorway>
walls                           ArrayList<Wall>
doorFrames                      ArrayList<DoorFrame>
itemObjects                     ArrayList<GameObject>
```

```
RoomManager
activeRoom                                                Room
rooms                                                     ArrayList<Room>
addDoorSprites()                                          void
addWallSprites()                                          void
generateRoom0 ()                                          void
generateComponents (ArrayList<ArrayList<GameObject>> , ArrayLi
getRoom (int)                                             Room
addNonDoor (ArrayList<ArrayList<GameObject>> , int, int)  void
moveToNextRoom (Doorway)                                  void
generateRoom1 ()                                          void
generateRooms ()                                          void
getDoor (int, int)                                        Door
generateRoom (ArrayList<Integer>, ArrayList<Integer>)     void
initializeCode(ArrayList<ArrayList<GameObject>>)          void
getWall(int, int)                                         Wall
generateRoom2 ()                                          void
setWallSprite(int, int, Wall)                             void
activeRoom                                                Room
activeRoomIndex                                           int
rooms                                                     ArrayList<Room>
```

Above is a clip from the IntelliJ created UML class Diagram, which is almost entirely incomprehensive, so we will just look at a small portion of it, specifically, the interactions between the Room class and the RoomManager class. The Room class is, as one might expect, the storage container of everything that the user (as a Player) walks around, sees, and interacts with. This includes Walls to run into, Doors to go through, NPCs to talk to, and GameObjects to puzzle with.

The RoomManager, on the other hand, does not store any of this data explicitly. Rather, it begins as empty before creating all of the Rooms demanded by the program. Additionally, it controls the contents of each room, including where the Walls, Doors, floors, and GameObjects actually appear. The codes that are input to the Room constructor (an ArrayList of ArrayLists) are generated here before actually calling the Room constructor. Finally, this allows for the Player’s position to be tracked within all of the Rooms without any of the individual Rooms being explicitly aware of any others.

Citations

We were assisted numerous times and in numerous ways by GetHub CoPilot, ChatGPT, and different entries on Stack Overflow.

https://chat.openai.com/
https://resources.github.com/copilot-for-business/?ef_id=_k_Cj0KCQiA67CrBhC1ARIsACKAa8QOnXebW6EXwmrUy8KhHfQB-2yr1KIhcT-V6UM9HhwIflagQH10HAAaAjRMEALw_wcB_k_&OCID=AIDcmmc3fhtaow_SEM__k_Cj0KCQiA67CrBhC1ARIsACKAa8QOnXebW6EXwmrUy8KhHfQB-2yr1KIhcT-V6UM9HhwIflagQH10HAAaAjRMEALw_wcB_k_&gad_source=1&gclid=Cj0KCQiA67CrBhC1ARIsACKAa8QOnXebW6EXwmrUy8KhHfQB-2yr1KIhcT-V6UM9HhwIflagQH10HAAaAjRMEALw_wcB
https://stackoverflow.com/