Software Requirements and Design Document

For

Group 10

Version 2.0

Authors:

Reece Gabbett

Ryan Beck

Parker Stone

Timur Bickbau

Marcos Sivira

1. Overview (5 points)

Development of the system is progressing at a healthy pace, with the fundamentals of world development (i.e., the MapEditorUI) and most of the necessary player elements (e.g., a health system) now in place. The final iteration will have our attention be focused on the development of the game world (overworld, dungeons, shops, etc.) using our MapEditorUI, adding a fun but challenging combat system, and improving our player system (stats, inventory, etc.) to allow interaction with the game world.

2. Functional Requirements (10 points)

Completed Requirements for Iteration 2:

1. The custom MapEditorUI: The map editor was arguably the highest-priority requirement that we completed in this iteration, as it now gives us the necessary tools to build the full world throughout the third iteration. The map editor currently includes functionality for map editing and defining the data for tiles in the tilesets. The editor can be enabled and disabled via toggle.
2. Basic collision: Lank does not walk through certain elements that he is not supposed to walk through (e.g., rocks, trees, and buildings). This was of medium priority and was a part of the preceding map editor requirement.
3. Health bar: There is now a permanent health bar (containing up to 6 hearts) on the top left of the screen. The health bar does not currently update as there are not yet any enemies who can damage Lank (current enemies do not deal damage). This was a high-priority requirement.
4. Basic enemy: A basic enemy functionality was implemented in which the enemy follows Lank when he is close and pushes him away when they touch him. Enemies can be killed by Lank's sword when hit. The functionality can be copied to other enemies so it can be expanded but it works well on its own for a lot of basic enemies.
5. Lank has been given hitboxes on his sword attacks, but not hurtboxes to take damage.

Future Requirements for Iteration 3:

1. High priority: Develop the world using the map editor.
2. High priority: Implement damage-dealing enemies.
3. High priority: Update the health bar to update when enemies deal damage.
4. High priority: Implement player warps.
5. High priority: Add enemy, warp, and item placement to the map editor.
6. High priority: Add ability to load enemy, boss, warp, and item placement/info from the map data file.
7. High priority: Finalize player state (stats other than health, such as defense and attack).

8. Medium priority: Implement healing items and health upgrades.
9. Medium priority: Implement an inventory system with items.
10. Medium priority: Implement saving functionality.
11. Low priority: Add economy (villages with traders and shops, and currency system to trade with).

3. Non-functional Requirements (10 points)

Scalability: Keep FPS locked to 60FPS for stability.

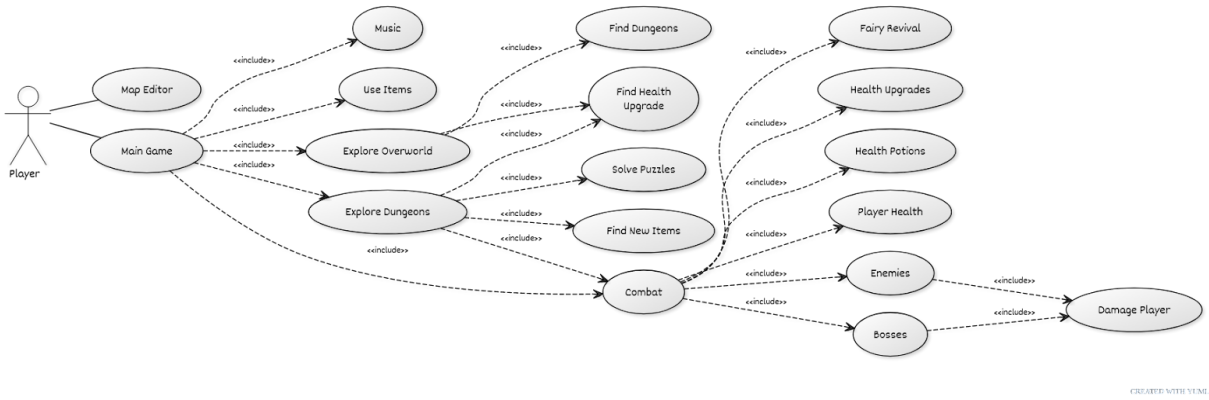
Data Integrity: Encrypt the saved data so that the user cannot compromise its integrity.

Usability: Run on Windows 7 or later, with keyboard and mouse.

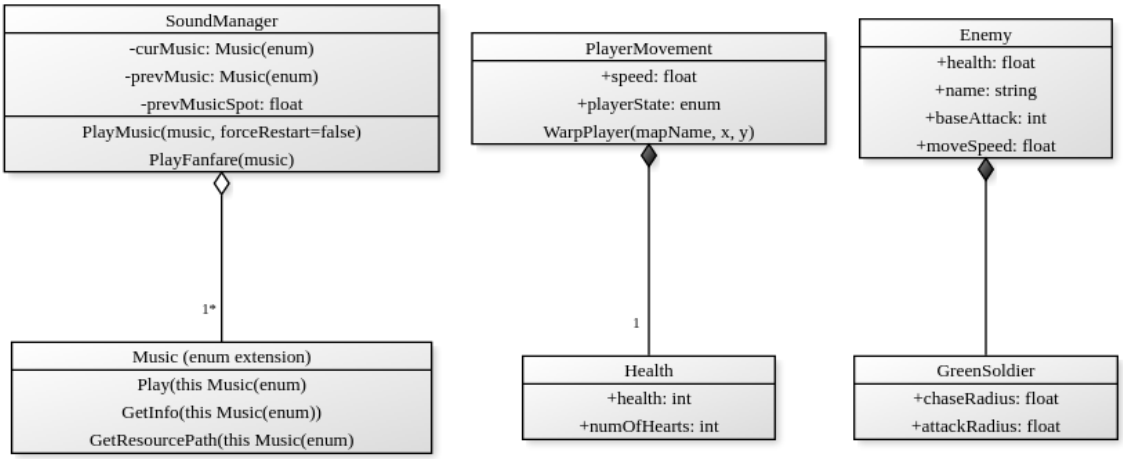
Reusability: Reuse assets (e.g., sprites and movements) for future development.

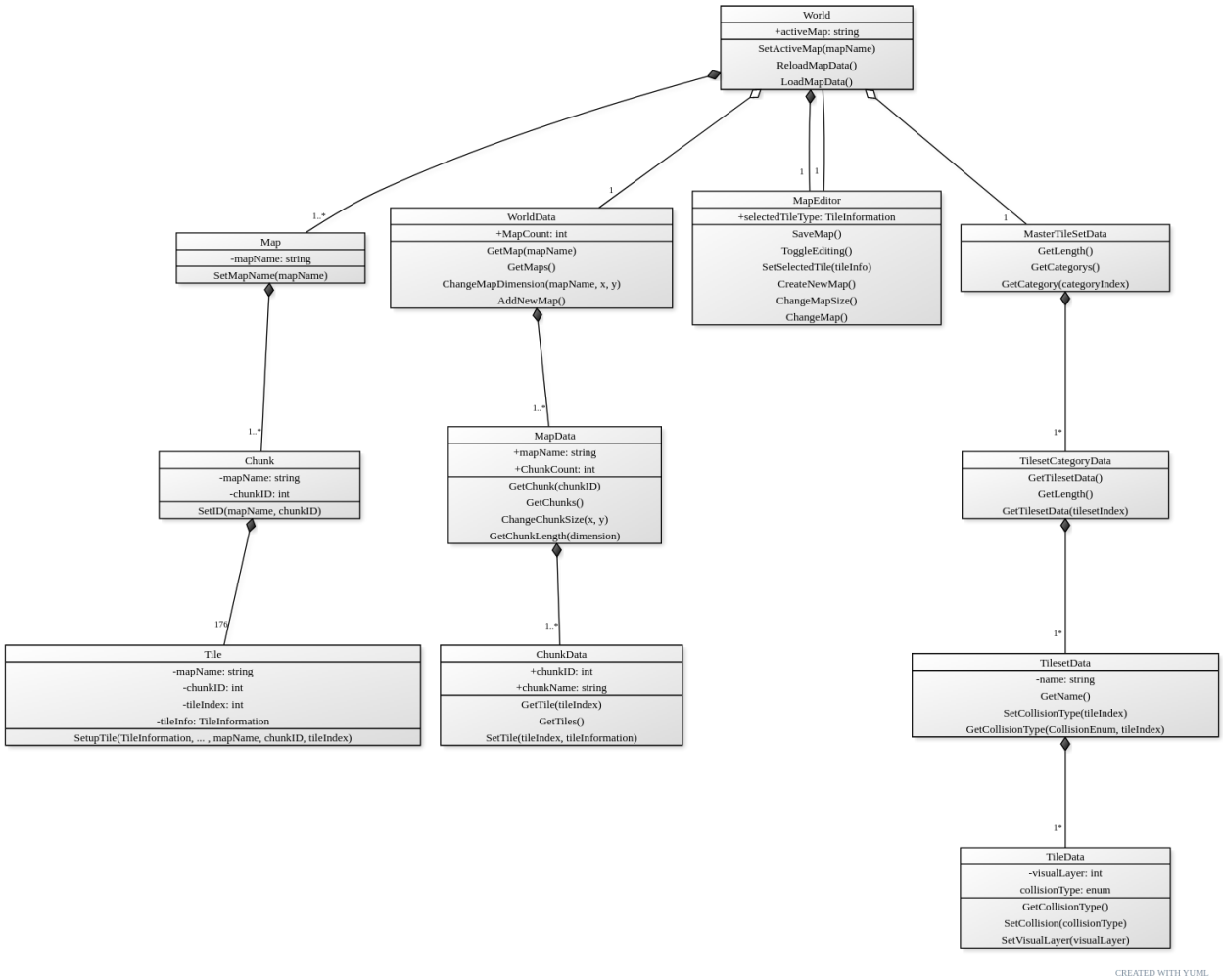
Reliability: Continue to ensure there are no game-breaking bugs.

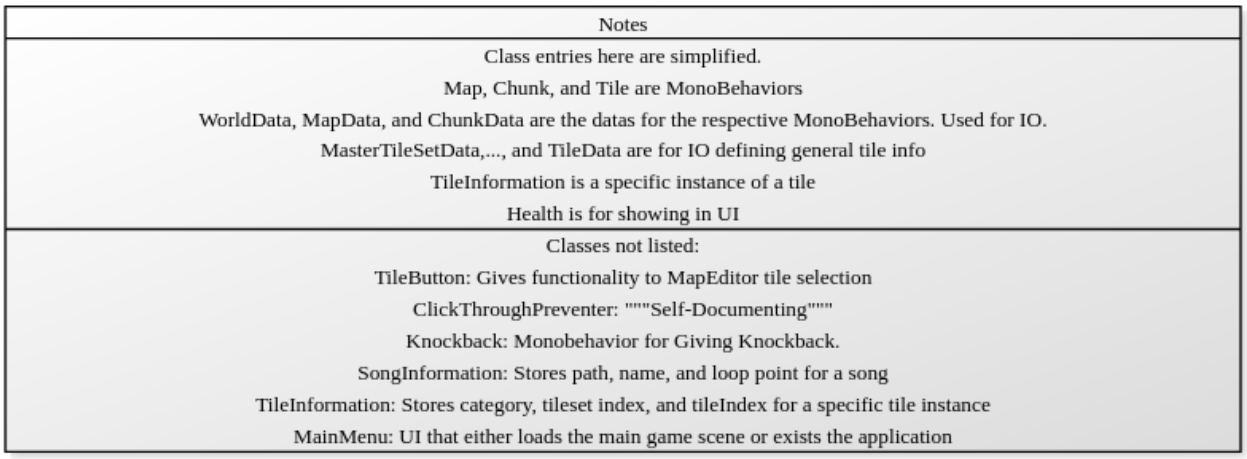
4. Use Case Diagram (10 points)



5. Class Diagram and/or Sequence Diagrams (15 points)


SoundManager
-curMusic: Music(enum)
-prevMusic: Music(enum)
-prevMusicSpot: float
PlayMusic(music, forceRestart=false)
PlayFanfare(music)
1*
Music (enum extension)
Play(this Music(enum)
GetInfo(this Music(enum))
GetResourcePath(this Music(enum)
PlayerMovement
+speed: float
+playerState: enum
WarpPlayer(mapName, x, y)
1
Health
+health: int
+numOfHearts: int
Enemy
+health: float
+name: string
+baseAttack: int
+moveSpeed: float
GreenSoldier
+chaseRadius: float
+attackRadius: float
CREATED WITH YUML


World
+activeMap: string
SetActiveMap(mapName)
ReloadMapData()
LoadMapData()
1..*
Map
-mapName: string
SetMapName(mapName)
1
WorldData
+MapCount: int
GetMap(mapName)
GetMaps()
ChangeMapDimension(mapName, x, y)
AddNewMap()
1
1
MapEditor
+selectedTileType: TileInformation
SaveMap()
ToggleEditing()
SetSelectedTile(tileInfo)
CreateNewMap()
ChangeMapSize()
ChangeMap()
1
MasterTileSetData
GetLength()
GetCategorys()
GetCategory(categoryIndex)
1..*
Chunk
-mapName: string
-chunkID: int
SetID(mapName, chunkID)
1..*
MapData
+mapName: string
+ChunkCount: int
GetChunk(chunkID)
GetChunks()
ChangeChunkSize(x, y)
GetChunkLength(dimension)
1*
TilesetCategoryData
GetTilesetData()
GetLength()
GetTilesetData(tilesetIndex)
176
Tile
-mapName: string
-chunkID: int
-tileIndex: int
-tileInfo: TileInformation
SetupTile(TileInformation, ... , mapName, chunkID, tileIndex)
1..*
ChunkData
+chunkID: int
+chunkName: string
GetTile(tileIndex)
GetTiles()
SetTile(tileIndex, tileInformation)
1*
TilesetData
-name: string
GetName()
SetCollisionType(tileIndex)
GetCollisionType(CollisionEnum, tileIndex)
1*
TileData
-visualLayer: int
collisionType: enum
GetCollisionType()
SetCollision(collisionType)
SetVisualLayer(visualLayer)
CREATED WITH YUML



6. Operating Environment (5 points)

The operating environment is a computer running Windows, using a keyboard and mouse control scheme. The Windows version must be compatible with Unity (i.e., it must be no older than Windows 7).

7. Assumptions and Dependencies (5 points)

It is assumed that the user has a keyboard and mouse, and will be running a version of Windows that is compatible with Unity (i.e., one that is no older than Windows 7).

The game will be dependent upon Cinemachine for cameras, and upon outsourced spritesheets for art design.