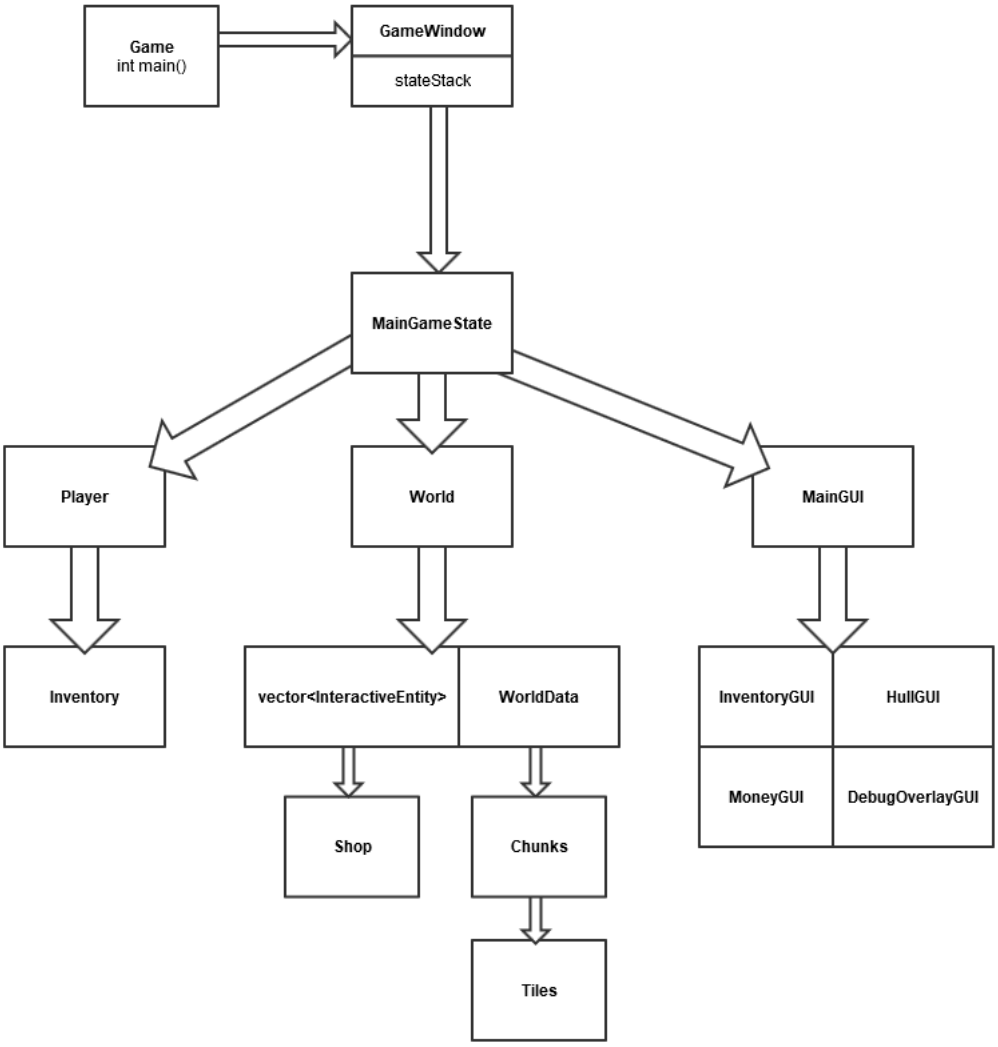
Project Diglett Code Overview

Introduction

This document provides an **informal, non-exhaustive** description of the code layout of Project Diglett. It is intended to give potential employers a good enough idea of the code's structure and design to be able to evaluate it usefully.



Basic structure

The `MainGameState` class represents the "main" state where the player is moving around and digging. On each frame, the `gameTick()` method is called to take inputs and update the game state, and the `draw()` method is called to draw the updated game state.

The class has a `Player` object representing the player's status (health, inventory, etc) and

containing the methods for moving the player.

There is also a `World` object containing information about the game world, and a `MainGUI` object containing drawable GUI panels to overlay the game world.

Input

The `MainInputHandler` class is defined within the `MainGameState` class and contains a `processInputs()` method that polls the window for input events and calls appropriate methods on `MainGameState` based on what input was received.

World

The `World` class contains a `WorldData` object encapsulating information about tiles in the game world. It contains a 2D array of `Chunk`s, where each `Chunk` is a 20×20 array of `Tile`s. The ambition behind the potentially peculiar design is to be able to implement an infinite world, but this feature hasn't been developed yet.

There are also `InteractiveEntity`s, of which there is currently only the `Shop`.

Entry point

The `main()` method is in `Game.cpp`. It creates a `GameWindow` object and passes control to the `mainLoop()` method in that object. `GameWindow` extends the `RenderWindow` class provided by the SFML library, which provides framerate control such that the main loop is repeated once every frame for as long as the game is running.

GameState stack

The `GameWindow` contains a stack of `GameState` objects, and calls `gameTick()` and `draw()` on the top state every tick. During normal play, the `MainGameState` will be active. Another state may be pushed if, for example, the player dies or the shop is opened. Pushing a new state lets us change the behaviour of the game, including how it handles inputs, as each `GameState` will have its own `InputHandler` specifying how inputs affect the game in that state.