

13/12/2017

BAALI Karim / ALGERA Pieter

Design Corp – IBO 1

**Hunger Maze**

Look at these fighters running for their life !

Table of contents

[Game introduction 2](#_Toc501022836)

[Maze generator 2](#_Toc501022837)

[Game logic 2](#_Toc501022838)

[Fighters 2](#_Toc501022839)

[Items 2](#_Toc501022840)

[GameMaster’s voice 3](#_Toc501022841)

[Used patterns 4](#_Toc501022842)

# Game introduction

They are brave! They are fast! They are … FIGHTERS! And they are going to have to find their way in our big maze to be able to win the game! A lot of items are available in the maze to help them to fight against other fighters, but nobody knows for how long these items are going to be in their inventories. Choose your favourite one, encourage him, happy Hunger Maze and may the odds be ever in your favour!

# Maze generator

First things first, the maze generator. We decided to implement a maze generator to make each launch of the game more interesting and give us the possibility to change the size of the maze during development. The algorithm that we implemented is called the Randomized Prim Algorithm. This algorithm creates a random spanning tree and so creates a perfect maze. We then take this perfect maze and break random walls to get a non-perfect maze so that the movements for the fighters are a little more interesting. The generation of the maze is divided into smaller chunks so that we can separate the generation into multiple threads and so have it go faster. To place the end point for the maze, we check along the right wall to find a spot where we could place the end point and break a hole through the wall. The generation is fairly quick and gives us very interesting results.

# Game logic

## Fighters

For each played game, 1% of the cells are occupied by fighters. Each of these fighters is represented by a “F” and is displayed in a distinct color on the maze. A fighter can move to the top, right, bottom or left cell to find a path leading to the exit of the maze. There is no “shorter path algorithm” in this project, they just move randomly on the maze with preferences for items or fighters depending on his strategy, but they never go back on the same path, unless there is no other path available then they will follow the path that they came with to find a new route to follow. If he finds an item on the maze, he keeps it in its inventory to use it later when attacking. Each item in its inventory increase its damages. There are two kinds of strategies when a fighter meets another one in the maze. He can attack him and try to kill him, or he can try to run away far from the enemy. When a fighter finds the exit, he is announced as the winner!

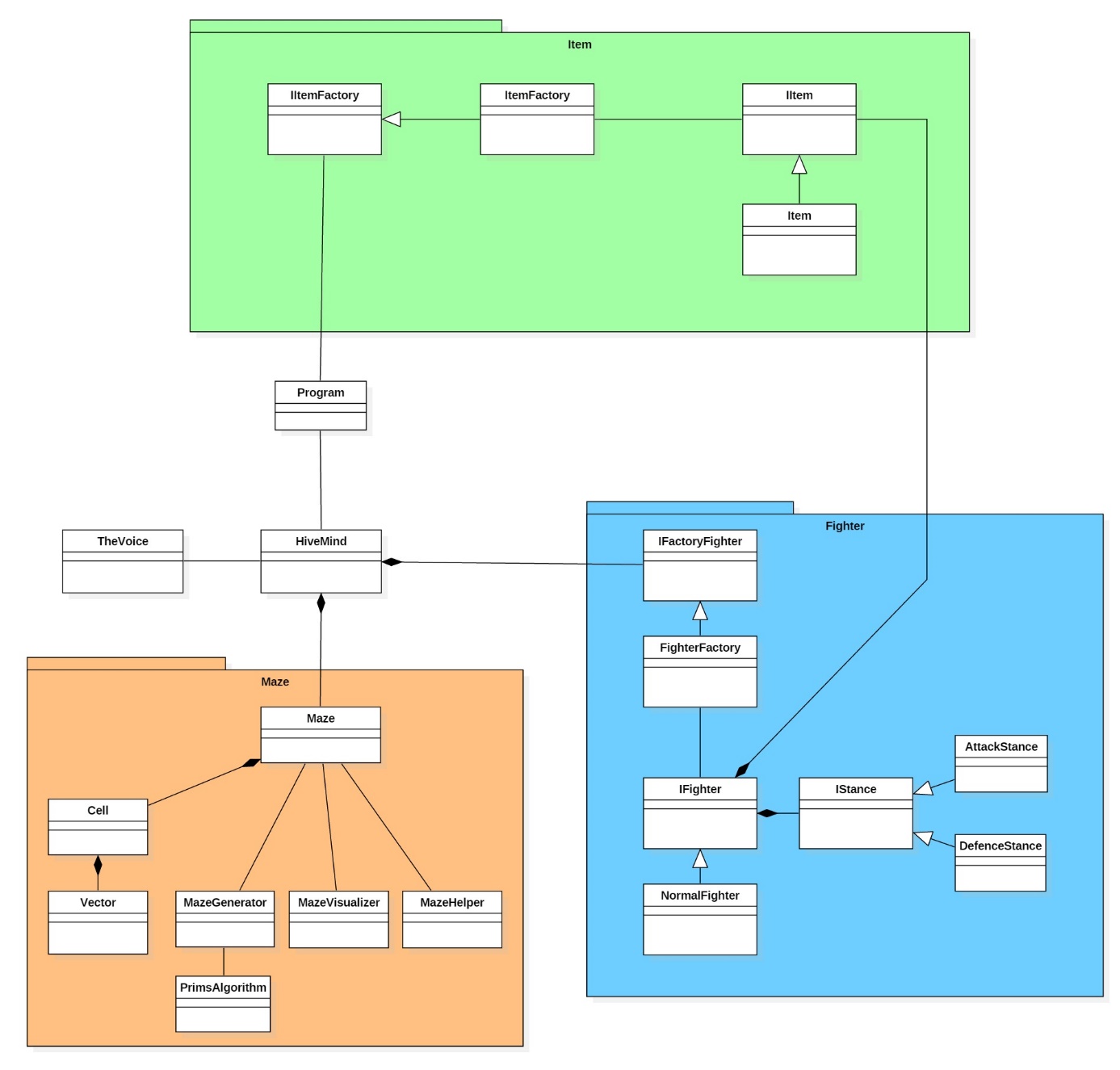
## Items

At the beginning of the game, a number of items equals to 10% of the total number of cells is used to generate all the items. They are placed on an unoccupied cell of the maze randomly chosen. An item has damage value, which is decreased each time a fighter uses it to attack. At the beginning, the items are in magenta on the maze, but the items which has been taken by a fighter before are displayed in blue after going back on a unoccupied cell.

## GameMaster’s voice

In a separate thread, a voice randomly announces to the fighters that they have lost all their items. The thread is sleeping for between 10 and 30 seconds each time, and all the items are going back on the maze. This voice is stopped when a fighter reaches the exit of the maze. The displayed message is chosen in a list of pre-fixed message to make it funnier. When a fighter reaches the exit of the maze, the thread is stopped.

# Used patterns



For this project, we used different kinds of design patterns to improve the code quality and maintainability of the project. The first one is the Factory pattern. We used it to generate different kinds of fighters at the beginning of the game, but also for the item generation. This way, we are able to create new fighters/items objects with distinct property just by passing a specific parameter to the factory. In each case, we have an interface IFactory, and a sub class Factory, which implement the first one, and where the generation logic is. These Factories doesn’t create a concrete object but an abstracted version (IFighter or IItem). The concrete objects implement those interfaces NormalFighter and Item). With this architecture, it will be easy for us to add new types of fighters or items and to generate the objects at the beginning.

The second pattern used in this project is the Strategy pattern. Each fighter can have two behaviours when he meets another fighter in the maze. If he has an item in its inventory, he can decide to attack the enemy, but if he doesn’t, he might prefer to run away without fighting. So, each fighter has an attribute IStance, with a react method, in which his behaviour is defined. If this object is an AttackStance (which implement the previous interface) then he is going to fight for his life no matter what, but if it’s a DefenseStance object, then he is going to run like never before !