Project Spectre Environment and Match Design

Table of Contents

[Overview 2](#_Toc410422970)

[Task Breakdown 2](#_Toc410422971)

[Terminology 3](#_Toc410422972)

[Entity 3](#_Toc410422973)

[Cell 3](#_Toc410422974)

[Coordinate 3](#_Toc410422975)

[Block 3](#_Toc410422976)

[Map Generation 3](#_Toc410422977)

[Descriptions 4](#_Toc410422978)

[Calculating Dimensions 4](#_Toc410422979)

[Defining Map Zones 4](#_Toc410422980)

[Placing the Boss 4](#_Toc410422981)

[Placing Players 4](#_Toc410422982)

[Generating Mini-Bosses 4](#_Toc410422983)

[Defining Placeholder Blocks 5](#_Toc410422984)

[Generating Enemies 5](#_Toc410422985)

[Generating Miscellaneous Elements 5](#_Toc410422986)

[Generating Tiles 5](#_Toc410422987)

[Pseudo Code 5](#_Toc410422988)

[Map Generation 6](#_Toc410422989)

[Entity Placement 11](#_Toc410422990)

[Abstract Map Elements 12](#_Toc410422991)

[Hazards 12](#_Toc410422992)

[Effects 13](#_Toc410422993)

[Round Procedure 13](#_Toc410422994)

[State Flow Diagram 13](#_Toc410422995)

# Overview

This document contains technical details, design work and task breakdowns for all areas of the game pertaining to the **game map** and **general match procedure**.

Game map design includes **map generation**, **entity creation**, **effects** and **hazards**.

Match procedure design includes state flow diagrams that walk through the general procedure of a match and illustrate some **key decisions** and actions to be taken as part of match procedure.

# Task Breakdown

This section contains a list of tasks that will be required for the completion of both environment implementation and match procedure. This list is not assumed to be wholly comprehensive, but instead aims to identify the major areas of work that need to be addressed.

* Entity placement
* Map generation
  + Create block class
  + Create block data structure
  + Determine zone layouts
    - Assign blocks to zones
  + Player placement
  + Mini-boss location selection
  + Mini-boss type selection
  + Placeholder block type selection
  + Enemy generation
    - Enemy group composition
  + Structure and object generation
* Dynamic tiling
  + Static tiles
  + Unique tiles
* Hazard spawning
  + Hazard spreading
  + Hazard ending
* Effect spawning
  + Effect ending

# Terminology

This document references a number of game-specific terms that are important for understanding the design properly. These terms are listed here with a brief explanation.

## Entity

An entity (in the context of this document) is considered to be any object that exists within the game world. This can be broken down into two categories: Blocking and Non-Blocking.

Blocking entities are objects such as the player, enemies, and environmental obstacles.

Non-Blocking entities are objects such as hazards, effects and abilities.

## Cell

A cell is a small area on the game map that an entity can occupy. An entity can occupy more than one cell at a time and can exist in inter-cell space. If a cell is occupied by a blocking entity, it cannot be entered by another blocking entity.

## Coordinate

A coordinate represents a specific location on the game map (with an origin of 0, 0 in the top left corner of the game map) that an entity can exist at, or be created at. Coordinates are smaller than cells, and can be used to reference specific places inside a cell.

## Block

A block is a small subdivision of the game world. Each block is composed of a number of cells and can be considered either placeholder or pre-set.

Placeholder blocks are filled with randomly-selected content at the beginning of each round and are scattered across the map.

Pre-set blocks will contain the same content across each round.

# Map Generation

Map generation occurs at the beginning of each new round. Maps will be generated using a combination of static and random elements. The steps associated with generation are listed in order below:

1. Calculate the map dimensions
2. Define map zones
3. Place the boss
4. Place the players
5. Place mini-bosses
6. Define placeholder blocks
7. Generate enemies
8. Generate miscellaneous objects
9. Generate tiles

## Descriptions

### Calculating Dimensions

Map dimensions are calculated in units of blocks. The map must be divisible by the size of a block in both dimensions, and must be comprised of an odd number of blocks in each dimension.

### Defining Map Zones

The game map is comprised of 3 distinct zones: Grass, Stone, and Arbiter. Each block in the game must be assigned to a zone before generation is done. This selection is done based on the overall size of the map, and relative sizes of each zone.

### Placing the Boss

The boss should be placed at the centre of the map, in the exact same location each round. The boss is contained within a pre-set block that has the same layout each round.

### Placing Players

At the start of a round, players should be placed in each of the four corners of the map. This selection is done completely randomly, starting with the player that was added to the player list first. No starting location should be more optimal than the others, and therefore it does not matter where players are placed.

### Generating Mini-Bosses

Mini-bosses should be placed in a spread-out way across the game world in such a way that they are relatively equidistant from all in-game players. Mini-boss generation is done by dynamically calculating how many should exist in each area of the map, and then selecting the best possible blocks for them.

### Defining Placeholder Blocks

Each placeholder block in the game world must have its contents defined before the round starts. The contents of a placeholder block are selected from a list of random options. When one of the options is selected, there is further opportunity to randomize the placement or type of content in the block.

### Generating Enemies

Enemies should be generated semi-randomly throughout the game world at the start of each round. Enemies are generated at the beginning of the round and not again, except in cases of deity abilities.

Generation should be done cell-by-cell, taking into consideration what currently exists in that cell (eg: mini-bosses, the boss, hazards). Enemies should be generated in groups of varying size depending on the enemy type.

### Generating Miscellaneous Elements

Any final objects placed in the game world, whether they can be interacted with or not, should be placed on a cell-by-cell basis and should be generated based on what type of item they are.

This process could be done via a sort of generation class, or a function within each object type that determines their generation process.

### Generating Tiles

Tile placement should be done after all cells have been placed, so that new structures and thematic locations receive the same tiling pass as the rest of the map. Tiling should be done in such a way that map edges, divisions and quadrants are automatically identified and tiled correctly. No tiling should need to be done by hand, except perhaps the base game map.

This should be completed by looping through each tile space on the game world, identifying where that space is in relation to the surrounding tiles and the overall game world, and placing the correct tile.

In cases where multiple tiles can be placed, a random selection should be made. This random selection should be more likely to choose nondescript tiles and less likely to pick unique tiles. An example of this is a grassy field where flower tiles are occasionally placed.

## Pseudo Code

### Map Generation

**Generate Map Function**

{

calculate the map dimensions

if map dimension calculation fails

{

exit

}

define map zones

place the boss

place the players

place mini-bosses

define placeholder blocks

generate enemies

generate miscellaneous objects

generate tiles

return the map data structure

}

**Calculate Map Dimensions Function**

{

if the map width is not divisible by the block width or

the map height is not divisible by the block height

{

print an error message

return false

}

if the map width is not even or the map height is not even

{

print an error message

return false

}

create a data structure for containing all map blocks

loop through and create each block, adding it to the data

structure

return true

}

**Define Map Zones Function**

{

Set the zone of the centre-most block to Arbiter Zone

loop through all blocks

{

if the block is within the innermost 40% of the map

{

if the block does not have a zone

{

set the zone of the block to Stone Zone

}

}

else

{

set the zone of the block to Grass Zone

}

}

}

**Place Boss Function**

{

Place the Arbiter at a random coordinate within the Arbiter Zone

set the selected block to be the Arbiter block

}

**Generate Mini-Bosses Function**

{

Place a random mini-boss in each of the four Stone Zone corner

blocks

if the Stone Zone is wider than 3 blocks and taller than 3 blocks

{

Place a random mini-boss in all Stone Zone edge blocks that

form a cross with the Arbiter Zone

}

place a random mini-boss in all Grass Zone edge blocks that form

a cross with the Arbiter Zone

set all selected blocks to be mini-boss blocks

}

**Define Placeholder Blocks Function**

{

loop through all untagged Grass Zone blocks

{

select a random block type from the Grass Zone list

set the block to the selected type

}

loop through all untagged stone blocks

{

select a random block type from the stone zone list

set the block to the selected type

}

}

**Generate Enemies Function**

{

loop through all enemy-type Grass Zone blocks

{

select a random Grass Zone enemy grouping

generate a number of enemies in the block across a random

range

}

loop through all enemy-type Stone Zone blocks

{

select a random Stone Zone enemy grouping

generate a number of enemies in the block across a random

range

}

}

**Select Enemy Grouping Function**

{

create an empty list of enemies

if the Zone is the Grass Zone

{

set the maximum block value to the Grass Zone Max

}

else

{

set the maximum block value to the Stone Zone Max

}

set the current value to the maximum block value

while the current value is greater than zero

{

select a random enemy with a value that is less than or

equal to the current value

add the enemy to the list

subtract the enemy value from the current value

}

return the enemy list

}

**Generate Objects Function**

{

loop through all special-type blocks

{

select a structure or object from the list of options

generate objects in the block across a random range

}

}

**Generate Tiles Function**

{

loop through all grass blocks

{

generate grass tiles based on edges and intersections

generate random unique tiles

}

loop through all stone blocks

{

generate stone tiles based on edges and intersection

generate random unique tiles

}

generate Arbiter tiles based on edges and intersections

generate random unique tiles

}

### Entity Placement

**Entity Placement Function**

{

If the entity is non-blocking

{

create an instance of the entity class

initialize the entity with the necessary values

set the entity's position to the desired x and y

coordinates

add the entity to the cell closest to the x and y

coordinates

}

else

{

if the closest cell to the x and y coordinates is empty

{

create an instance of the entity class

initialize the entity with the necessary values

set the entity's position to the desired x and y

coordinates

add the entity to the cell closest to the x and y

coordinates

}

else

{

return false

}

}

return true

}

# Abstract Map Elements

Abstract map elements are considered to be Non-Blocking entities that have an effect on the game world and other entities. The major categories of these elements are **Hazards** and **Effects**.

## Hazards

Hazards are entities that can be assigned to a specific block or area of the map. Hazards may or may not be able to spread, but are identified by their ability to harm other entities in the game.

## Effects

Effects are entities that are generated as a result of actions taken in the game by players, artificial intelligence-driven entities and hazards. These are generally defined by their lack of interaction with other entities, and the fact that they are primarily animations or particle systems that disappear after completion.

# Round Procedure

Each round follows a distinct procedure from start to finish. This process is defined below using a state flow diagram that illustrates how a player might progress through the game round-by-round.

It is assumed that the player’s game can end at any point due to them exiting or due to the server closing.

## State Flow Diagram

