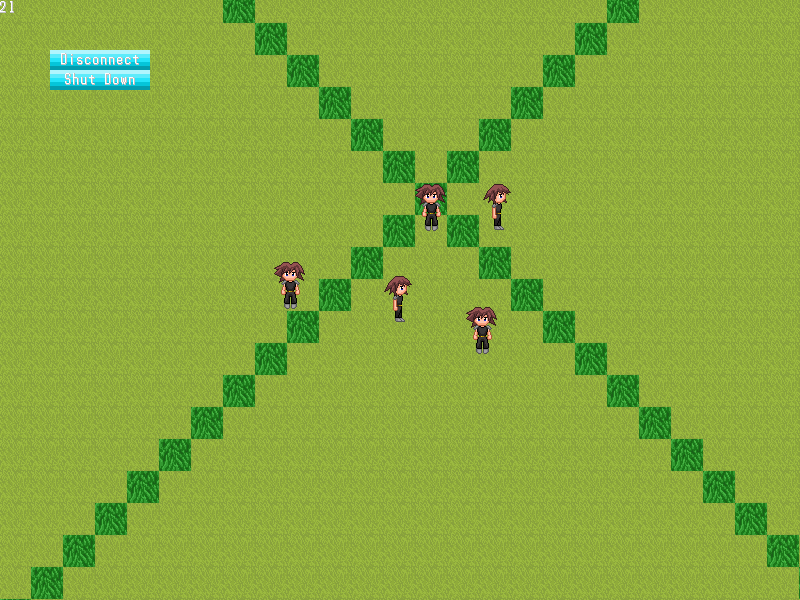
Tortuga Game Design Document

By Kayne Ruse, KR Game Studios



# Table of Contents

Table of Contents 2

Introduction 3

Elevator Pitch 3

Monetization 3

My Greenlight Checklist 4

Description 4

Required Upkeep 4

Gameplay Overview 5

The Game Map (TODO) 6

Combat Mechanics (TODO) 8

Enemy Types (TODO) 9

Item Types (TODO) 10

Server Management (TODO) 11

Development Resources 12

Platforms 12

Languages and APIs 12

Map System 13

Region 13

RegionPager 13

Allocators 13

File Format Functors 13

Communication Protocols 14

Lua Scripting API 15

Map API 15

Combat API (TODO) 15

Enemy API (TODO) 15

Modding Support (TODO) 16

To Do List 17

Possibilities 17

Additions to the GDD 17

References 17

Random Thoughts 17

# Introduction

This is the game design document for Tortuga. This is intended to plan out required tasks for the game, as well as serve as an instructional text for modders and others interested in Tortuga’s development cycle.

I have a long road ahead of me, and design docs are never really finished until the project is. So, if you see any italic text, you can consider that to be an incomplete or removed section of text, or an inline comment.

## Elevator Pitch

Tortuga is a 2D multiplayer JRPG featuring permadeath (deletion of a character upon death). The emphasis of this game is on multiplayer cooperation, exploration and customization. The game runs on customizable server software that can support up to 150 simultaneous players or more.

This game is inspired by classic 2D RPGs, as well as more modern sandbox MMOs. This project is currently independently created and funded, with the goal of creating a game that will engage user's imagination and inspire a large modding community.

## Monetization

I’m currently planning to submit this game to the Steam Greenlight process, and offering the following pricing system.

* Free Until Alpha
* $5 During Alpha
* $10 During Beta
* $15 On Release

Anybody who has purchased this game has access to all of the development files, as well as a license to modify and redistribute the game to anybody else who has also purchased the “vanilla” (unmodified) version. This is to encourage mods and customisation of the game.

I will probably start using a modified version of the Uplink Developer’s License if and when this game becomes available on Steam.

# My Greenlight Checklist

## Description

* Early Access
* Multiplayer / Private Server MMO
* Stat Heavy JRPG
* Procedurally Generated Open World
* Dungeon Diving
* Permadeath
* Highly Moddable
* Source Code And Resources Available On Purchase

## Required Upkeep

* Change the license (probably the Uplink license).
* Keep the documentation up to date (The GDD)
* Promotional video (first impressions)
* Developer video (more in depth view of the game)
* Screenshot mock-ups (clearly marked as such)

# Gameplay Overview

The game takes place in a procedurally generated world, consisting of a single “overworld”, and a series of “dungeons”. Different areas of the overworld have different difficulty levels, which scale the types and strengths of the enemies that are found there, and subsequently the rewards for defeating them. These difficulty levels also extend to the dungeons, whose entrances are found scattered around the overworld.

Each player can create a set of player characters (PCs) that are connected to their account on the server; however they can only play as one at a time. Each character has standard metadata, including name, sprite, etc. that make that character unique. In addition to this, each character also has a set of statistics (used for combat purposes and progression) and an inventory (their personal items). Each PC’s stats can be increased either by levelling up, or by using certain items.

Items can be found throughout the world, either dropped by enemies or hidden in chests. Item types include consumables (food that increases HP, boosts, stats, etc.) equipment (for boosting stats), legendary (unique and hard to find) and mundane (vendor trash, etc). Equipment items can be worn by players to boost certain stats; each piece of equipment has a certain “type” which dictates what slot it can be used in.

If a player’s character runs out of health, and they have no way to save themselves from death, then that character is deleted; this is called permadeath. Any items that the character is holding at the time are also lost (with the possible exception of legendary items).

A combat portal represents a fight in progress, located in the overworld or a dungeon. Random portals can be generated near players, sucking them and their parties in against their will. Static portals can be generated by the dungeons, as a type of boss or mini-boss encounter. Portals have visual cues, signifying their content (such as difficulty, current participants, etc.). A player can enter an existing battle, or flee one if the situation turns dire.

Enemies are procedurally generated when a combat portal is first created, based on the environment (i.e. plant-based monsters would only live in a forest) of the portal and the difficulty dictated by the location. Defeating enemies earns players rewards, but not all enemies in a portal need to be defeated to receive that reward. Landing a killing blow against an enemy will grant that player a bonus reward, while killing the last enemy on a portal will grant a significantly larger prize.

The overall tone of the game is geared towards player cooperation in order to progress, but this is not explicitly required. Players can progress perfectly well alone, and they can even “grief” other players; some features in the future may actually support and reward these opposing styles of play.

# The Game Map (TODO)

*The game’s maps are for the most part procedurally generated. The main world map that connects most of the game world together is called the overworld. The centre of the overworld map (spawning around (0, 0)) is the root town, where players first enter the server. Safe towns, outposts, etc. like this root town are also generated elsewhere, depending on the overworld’s generation algorithm.*

*overworld title*

*The overworld consists of different biomes including mountains, oceans, plains, etc. Different sections are also given difficulty ratings, from zero (for towns, safe zones, etc.) upwards. The difficulty ratings are fixed, meaning that you can be easily outgunned in a dangerous area, so it’s a good idea to keep track of the ratings as you explore.*

*Dungeons*

*Scattered around the overworld are several types of “dungeons”. Unique monsters and items can be found inside, and they usually have one final “boss” monster that drops the biggest reward. Each dungeon has a specific difficulty rating, based partially on its location.*

*Each area inside a dungeon is procedurally generated, based on various factors including the dungeon’s type, difficulty rating, how deep the player is, and where in the world the dungeon is located. The locations of these dungeons are generated by the overworld algorithm; there are also unique “legendary” dungeons that are guaranteed to spawn with certain parameters.*

*Ruins*

*The ruins are the remains of an ancient civilization. Their structures have multiple levels, with sprawling corridors and halls honeycombing through the ground.*

*Towers*

*Towers are gauntlet-style challenges that get progressively harder the higher you go. Each level in a tower is a simple room, but it has its own monsters and challenges.*

*Forests*

*Forests take up actual space in the overworld, but they do exist in their own instance. They’re often bigger on the inside than the outside, and often have multiple exits. The Forests are single levels, with a mazelike structure.*

*Caves*

*Caves have several exits, often extremely far apart from each other with the cave acting as a fast transit system. However, like other dungeons, caves also have dangerous obstacles, and must be explored to reach the other end. Their structures can be segmented into separate chambers, each with unique features.*

*Legendary Dungeons*

*Legendary dungeons are special cases in the generation algorithm. These unique locations are guaranteed to spawn with certain features in every server, and are often related to the plot or the game as a whole.*

# Combat Mechanics (TODO)

*• Level*

*• Experience*

*• Maximum Health*

*• Health*

*• Maximum Mana*

*• Mana*

*• Attack*

*• Defence*

*• Intelligence*

*• Resistance*

*• Accuracy*

*• Evasion*

*• Luck*

*Combat Mechanics*

*Tortuga’s combat consists of a unique drop-in/drop-out multiplayer active time battle system, inspired by earlier Final Fantasy instalments. Battles that a player can participate in will be visible in the game world as “combat portals”. See the next section for more information.*

*The combat system will allow several people to fight side by side as a group. Each player will be able to choose their own actions, including attacking with their equipped weapons, using spells and items, or fleeing the battle, etc. To prevent a player from taking too many actions too fast, that player must wait for their ATB gauge to fill completely before acting, after which it resets to zero. The speed at which a character’s ATB gauge refills is affected by that character’s speed stat; each player needs 100 “ticks” in their ATB gauge to attack. They can select their specific attack from the menus while their meter is loading, but actions won’t be taken until the meter is full.*

*For a speed of 5, it would take 20 seconds to take an action, while for a speed of 50, only 2 seconds are taken, etc. These extreme time delays may seem strange, but I have a hunch that it may be necessary to allow players to search their menus. I’ll make tweaks further down the road.*

*TODO: Stats and equations*

*Since permadeath is such a massive aspect of the game, a player must have the ability to flee a battle at any time. Fleeing a battle causes a player to take a penalty such as losing money, experience points, or more. Leaving a battle, if there are still players fighting that battle, does not end the battle itself. Instead, when all monsters inside a combat portal have been defeated, the portal disappears and the remaining players are rewarded. However, if there are no players currently fighting a particular battle and the monsters were not defeated, the battle resets; the monsters regain their full health and are cured of any status ailments.*

*The complement of the ability to flee a battle is the ability to join a battle in progress. If there are players currently inside a combat portal, a player within sight of the portal will be able to distinguish this. In addition, that player can join the battle to assist the other players, or to possibly cause those players problems (like stealing the loot).*

# Enemy Types (TODO)

*TODO*

*So I'm thinking about Boss Battles for the game. The game will have a drop in-drop out battle system, where if at least one person is still fighting the battle continues, but the people who drop out usually give something up in return (i.e. no exp, lose exp or gold, etc.) This will make permadeath easier on the players.*

*Sidenote: If a player does defeat a minion, or a part of a monster horde, they should receive at least part of the reward.*

*Anyway, for Boss battles, I want to encourage several people to take on a boss at once. one way to do this is to make the boss super strong, but I don't really want to rely on that. Another idea was to have several "minion" creatures that the boss uses.*

*i.e. You're fighting the Frog King, who summons four Frog Knights at the beginning of the battle. Every time a Frog Knight dies, a new one is summoned, so you're never fighting less than five creatures at once. The battle ends when the Frog King is dead, but the king never attacks you directly, instead summoning new Frog Knights, as well as healing and buffing the knights already on the field.*

*To defeat the Frog King, you can't just focus on him, since letting the Frog Knights attack you will almost always result in your death. So to defeat the knights and the king, the best strategy is to stun, incapacitate or draw the attention of the knights to other players, while one person attacks the king directly.*

*Now, although this would take a lot of mid level players to bring down, I could scale the number of knights being summoned in addition to the stats of the creatures for when there's only one or two players in the fight. With the drop in-drop out mechanic, I can scale the combat to match the number of players as well.*

*Anyway, this is ages away. I just wish development could go faster than it is right now. Making a video game is hard work, doubly so when you're doing it solo.*

*MOVED: Monsters*

*Monsters are script driven opponents generated inside combat portals, which also drop random rewards when they are killed. At this stage, there are no concrete plans for monsters outside of this.*

*TODO: Types of monsters, and their possible algorithms.*

# Item Types (TODO)

*TODO*

*Items*

*Items are randomly found throughout the world, or dropped by defeated monsters and players. Item types include consumables (food that increases HP, etc.) equipment (for boosting stats), legendary (unique and hard to find) and mundane (vendor trash, etc).*

*TODO: legendary items*

*Equipment*

*Equipment items are items that can be worn by players to boost certain stats. Each piece of* equipment has a certain “type” which dictates what slot it can be used in.

*TODO: Expand these few sections when I’ve reached that part of development.*

# Server Management (TODO)

*Server Mechanics*

*TODO: What can a server do, and how does it do it?*

*Players can setup their own game server, while other players can connect via the internet or over a LAN. The first time a player connects to a server, they have to create a new character for that specific server.*

*Server moderators can whitelist or blacklist players, as well as install mods. Mods are packaged in zipped archives, and dropped into the server’s “mods” folder (simplicity here is what I’m aiming for). Actually creating these mods requires knowledge of the scripting API, as well as familiarity with the lua programming language.*

*At first, only the world generation code can be modified, but eventually new graphics, etc. will be available. More in-depth modding is possible by altering the source code, but this would require obtaining the new programs. For security reasons, the vanilla client will not be allowed to connect to servers with modified source code, and the vanilla server will reject modified clients.*

*Data Storage*

*TODO*

*User Accounts*

*TODO*

*The player accounts can be handled using Amazon servers down the line, but for now players have to input their username into the configuration file. This username is used by the server, so if two people connect using the same username, there could be problems.*

*Each person who accesses a server must have their own user account. This allows players to keep track of their PCs, items, and other settings. This will also allow a server owner to whitelist or blacklist certain players, as well as other server specific options.*

*Each user account will have a certain number of PC slots. The items, etc. that a character collects stays with that character when a user logs out.*

*The accounts will be stored in a database.*

*Lua Scripting*

*TODO: Information on the packaged scripts and the APIs.*

*SQL Scripting*

*TODO*

# Development Resources

From this section onwards, this document focuses on the development side of Tortuga. There’s a lot more technobabble, intended for modders and such. This is also the primary documentation for the various components that make up the games codebase.

## Platforms

At this stage, due to a limited scope and budget, this game will only be available on Windows. Hopefully, I might be able to port it to Linux and Mac eventually.

*TODO: Upgrading to SDL 2.0 would be a notable advantage*

## Languages and APIs

The languages of choice for creating this game are C++11 and lua, for their large user bases and wide feature sets. Third party libraries I’m using include SDL (Simple DirectMedia Layer), SDL\_net and SQLite3, for much the same reasons.

# Map System

Tortuga’s map system is designed to allow a pseudo-infinite 2D tile-based map. As a player reaches the edges of the map, the map is expands to accommodate them, giving the impression of an infinitely large world. The components of the map system are detailed below.

There is no built in code for generating the world, instead there are template parameters available to pass each region to a lua script, or you can hard code a new parameter class. See the [scripting section](#_Map_API) for more information.

*TODO: Should I include the tile sheet as part of the map system?*

## Region

The game’s map is divided up into units called “Regions”. The Region class is fairly simple, requiring the X and Y positions on creation, and provides two functions for accessing the three dimensional array of tiles. The array’s size is defined by three macros: REGION\_WIDTH, REGION\_HEIGHT and REGION\_DEPTH.

This class also defines the type Region::type\_t (usually unsigned short) which is the internal storage type for the tiles. The tiles are not zero indexed; instead 0 represents an empty tile.

## RegionPager

The paging system is divided into two parts: RegionPagerBase and RegionPager. The abstract base class handles the list of Region objects, and provides wrapper functions for tile access. The base class actively uses the interface methods provided, so these must be defined in the derived class.

The derived class, which takes two template parameters, doesn’t do much itself, apart from bridging the gap between the base class and the functor classes provided as template parameters.

## Allocators

There are currently two allocator classes, BlankAllocator and LuaAllocator. The first creates and cleans up a region object, and simply leaves the default values in place. The second also passes the object to a designated lua function for processing both after creating it and before freeing it.

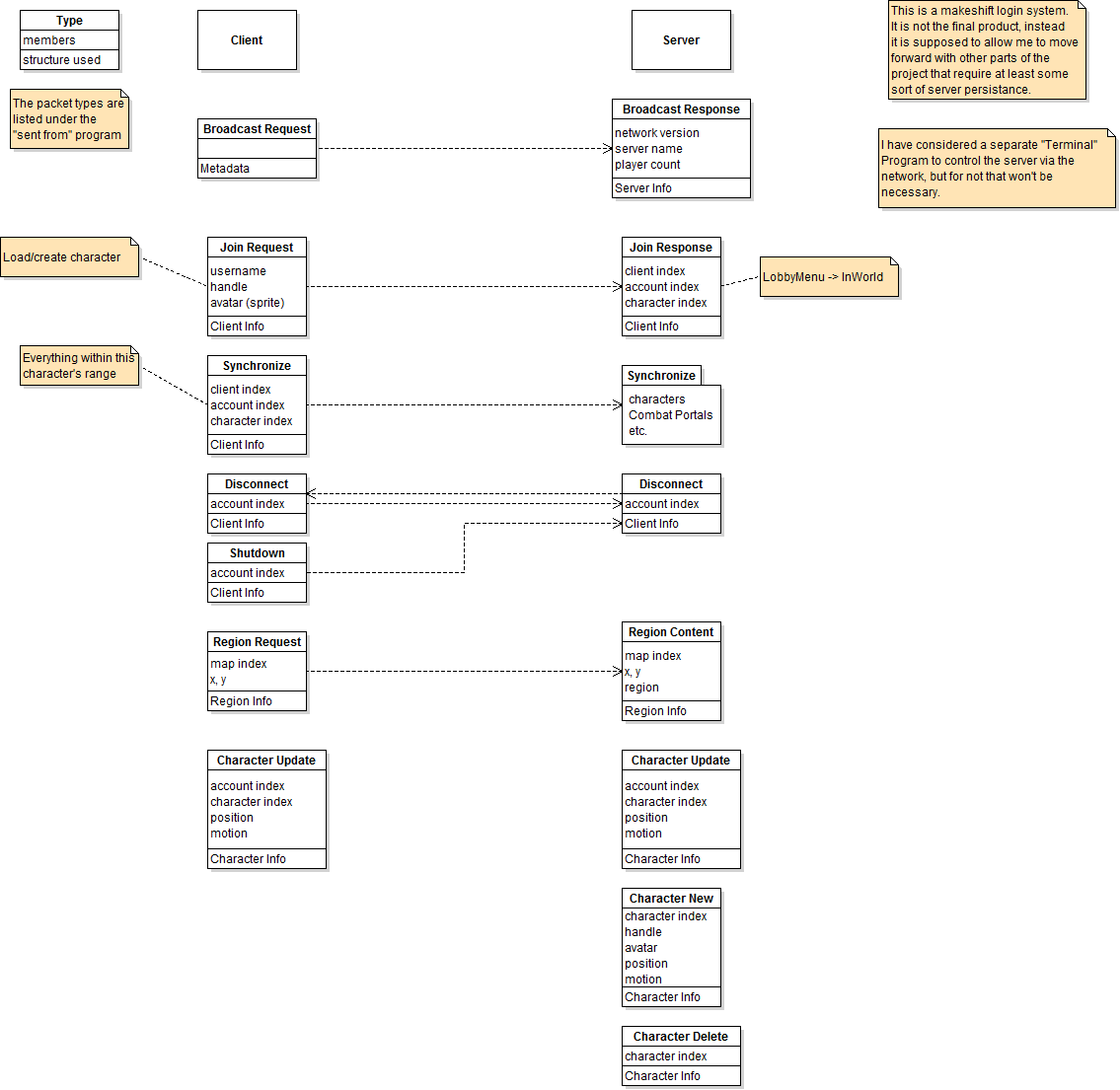
## File Format Functors

The format functors require that the program provide the file path to save the map in before it is used; otherwise its behaviour is undefined. The first, DummyFormat, does literally nothing. The second, LuaFormat, will provide an existing object to lua’s save function, or create a temporary Region, and provide that to lua’s load function. Please note that if the load function returns false, then the specified file was not found and the temporary object is freed instead of being passed to the pager.

# Communication Protocols

Communication between the client and server is achieved by UDP. All packets are first serialized into a buffer, before being sent. The serialization processes are customized for each member of the class “SerialPacket”, and stored on serial.cpp.

The current process is a stopgap measure until security procedures can be implemented. However, it is likely that a large amount of it will carry over to the final version. This diagram will also be amended when other features such as combat are implemented.



*TODO: More details*

# Lua Scripting API

The server supports modification partially via external scripting.

## Map API

*TODO: When I start using multiple maps at once, this will need rewriting*

The map API is referenced using the library name “map”. Below is a list of the available functions for manipulating either regions in the map, or the map as a whole. The first four functions are only called by the map system when the provided lua integrated template parameters are used.

It should be noted that when settile() and gettile() are operating on a region, the tiles **are not** zero indexed (to conform to lua conventions), but when it is operating on the entire map, then they **are** zero indexed. Therefore, region (0, 0)’s tile (1, 1, 1) is also the universal tile (0, 0, 0).

|  |  |
| --- | --- |
| create(region) | This is called on a newly created region. This function should be overwritten by custom code. |
| unload(region) | This is called on a region just before being unloaded from memory. This function should be overwritten by custom code. |
| load(region, dir) | This is called on a newly created region object, which should be filled with data from the disk. Returns true if a file was loaded, otherwise returns false. This function should be overwritten by custom code. |
| save(region, dir) | This is called on a region, which should be saved to the disk. This function should be overwritten by custom code. |
| settile([r,] x, y, z, v) | If a region is provided for r, then that region’s tile at (x, y, z) is set to the value v. Otherwise, the tile at the global position (x, y, z) is set to the value v. |
| gettile([r,] x, y, z) | If a region is provided for r, then that region’s tile at (x, y, z) returned. Otherwise, the tile at the global position (x, y, z) is returned. |
| getx(region) | Get the x position of the leftmost column of tiles in this region. |
| gety(region) | Get the y position of the topmost row of tiles in this region. |
| getregionwidth() | Returns the internal value for REGION\_WIDTH. |
| getregionheight() | Returns the internal value for REGION\_HEIGHT. |
| getregondepth() | Returns the internal value for REGION\_DEPTH. |

## Combat API (TODO)

*TODO*

## Enemy API (TODO)

*TODO*

# Modding Support (TODO)

*TODO*

* *An unmodified client must be able to connect to any server, be it vanilla, custom or modded.*
* *A server must allow the client to operate correctly.*
* *If any information causes issues, it must be resolved.*
* *If the source of the issue is the vanilla software, than that is the developer's responsibility to bring the software into line with the expected features.*
* *If however, the problem is caused by modified software or assets, than it is the responsibility of the server operator to resolve the issue.*

# To Do List

This section is mostly just rubbish dumped here, that will eventually be moved to a previous part of the document, or outright deleted.

## Possibilities

* NPC Characters
* Factional Warfare
* Controllable Mana Nodes, captured by FW, similar to Meridian 59
* Chat system
* Money to acquire items from NPC shops
* Chaos Gates for long distance teleportation
* Gates could be automatic, but only activate if you’ve reached that gate

## Additions to the GDD

* Add the combat and statistic equations
* Forest enemies tend to be poisonous
* Tower enemies are usually difficult, and humanoid

## References

* [Fewer Options, More Meaningful Choices](•%09http:/www.gamasutra.com/blogs/EvanJones/20130701/195361/Fewer_Options_More_Meaningful_Choices.php) - Gamasutra

## Random Thoughts

I read in an article about magic being affected by the environment. What if in the ATB system, magic was affected by what type of terrain you were fighting on? Like, if you were in a volcano, fire magic was more powerful, but ice magic was weaker? Unless you had an ice crystal to draw magic from, which makes up for the lack of ice magic sources, but the crystals only had limited amount of uses?

When a combat instance is created, a list of enemies is generated based on a series of parameters, like the biome/terrain type/dungeon type/etc. and the level (base and derived) of the area, combat instance, opponents, etc. I kinda… yeah, I’ll have fixed levelling, so your character can be outgunned by an OP boss monster, so I won’t need to adjust the difficulty based on the human characters in the fight (thank god, that would be complex has hell).