Tortuga Game Design Document

By Kayne Ruse

Introduction 2

Elevator Pitch 2

Monetization 2

Gameplay 2

Movement 2

Combat Instances 2

Combat Mechanics 3

Permadeath 3

Monsters 3

Items 3

Equipment 3

The Game Map 4

Dungeons 4

Ruins 4

Towers 4

Forests 4

Caves 4

Legendary Dungeons 4

Navigating the Client 5

Server Management 5

Server Mechanics 5

Player Accounts 5

Player Characters 5

Player Character Statistics 6

User Accounts 6

Lua Scripting 6

SQL Scripting 6

Todo List 6

Development Resources 7

Languages and APIs 7

Map Structure 7

Region 7

RegionPager 7

Generator Functors 7

Format Functors 7

Lua’s Region API 8

Modding Support 8

Communication Protocols 8

Platforms 8

# Introduction

This is the 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 an open source 2D multiplayer rogue like RPG. The game runs on customizable player run servers, which can hold at least 150 unique players. This project is currently independently created and funded, with the goal of releasing an early version for alpha funding and incorporating community feedback.

## Monetization

I’d like to develop this game as an open source project until I reach alpha, at which point I’ll run a kickstarter and begin taking preorders. The preorder system will work just like Minecraft’s preorders: players can buy the game during alpha for 50% of the initial price, or 75% during beta. The kickstarter will be used partially to raise awareness of this project at the beginning of the alpha.

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.

# Gameplay

## Movement Mechanics

The game is a top down RPG, so the players will walk around the 2D world, while being unable to pass through solid objects (including walls, water, etc). The player can enter/exit areas by colliding with door objects. These different areas include buildings, dungeons and combat instances. There will also be the possibility of teleportation at some point.

## Combat Mechanics

Tortuga’s combat consists of a unique drop-in/drop-out multiplayer active time battle system. Battles that a player can participate in will be visible in the game world as “combat portals”.

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 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 taking an action, after which it resets to zero. The speed at which a character’s ATB gauge refills is affected by that character’s statistics.

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. When all of the monsters inside a combat portal have been defeated, the portal disappears and the 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).

*fleeing and joining a battle need to be moved to the next section.*

## Combat Portals

Combat takes place in specialized instanced rooms, known as combat portals. The combat mechanics are reminiscent of early Final Fantasy instalments, with two sided combat using an active time battle system.

Each combat instance is created in the overworld or a dungeon at random, and is located where the players encountered it until that instance is complete. Players can also create combat instances purposefully by attacking another player; this pits these two players against each other.

Players are capable of entering pre-existing combat instances from the game world (via combat portals), and individual players already in combat can exit that combat (by selecting the run command).

## Permadeath

One of the most notable features of the game is the presence of permadeath: deletion of a character upon death. The purpose of this feature is twofold. First, to add an extra layer of challenge to the game, and secondly, to prevent any one character from becoming too powerful without taking risks. Any legendary items that the character owns at the time of death are returned to the world.

One of Tortuga’s most influential game mechanics is permadeath i.e. the deletion of a character when the player runs out of life.

## Monsters

The monsters are, at this stage, just generic programming placeholders. However, they do have rudimentary AI, as well as full combat statistics for basic PvE play. When a natural combat portal is spawned in the overworld, monsters based on that specific location are selected, and created inside that instance.

## 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) and mundane (vendor trash, etc).

## 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.

# The Game Map

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.

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 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 are 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.

# Navigating the Client

TODO: menus, user interface, etc.

# Server Management

## Server Mechanics

TODO

What can a server do, and how does it do it?

Server Structure

TODO

Data Storage

TODO

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 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 if the server’s code is modified, rather than just the scripts.

## Player Accounts

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.

## Player Characters

Each player can create a set of player characters that are connected to their account on the server. 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).

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).

When a player first logs onto a certain server, they must create a new character. Characters generally cannot be transferred from one server to another, preventing players from importing overpowered characters into a world that isn’t ready for them.

The player characters (PCs) will be created and customized by users. The PCs will gain levels and stat increases as the players progress with that character. When a character’s health value reaches zero, that character will die and is deleted from the server (see permadeath).

## Player Character Statistics

Each PC has their own unique set of statistics (stats). Possible PC stats include: TODO

# User Accounts

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

Servers can run custom scripts on the clients, but there needs to be a limit to this.

# SQL Scripting

TODO

# Todo List

Clean up this document

Page breaks

Add more

*non-player characters? If I include a storyline further down the road, NPCs would be important.*

*factional warefare? PvP is probably the best focus for this game. Creating guilds, parties, etc. would be a good way to encourage communities on the same server. This might be a better fit for a single shard game.*

*at least a basic chat system is needed at some point*

*money & economy? again, something like this probably wouldn’t fit with a game aiming for 150 people per server.*

*mana nodes? If magic becomes a gameplay mechanic, mana nodes that increase that magic would be useful. Also, faction controlled mana nodes would be pretty cool. It seemed to work for Maridian 59.*

*world gates? Long distance teleportation. For root towns, it could be automatic, but maybe you need to have found other gate to be able to teleport there.*

*This needs some serious editing for consistency*

# Development Resources

## 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 Structure

## Region

The game’s map is divided up into units called “Regions”, which are stored in the Region class. This class is fairly simple, requiring the width, height, depth, X and Y positions on creation, and provides two functions for accessing the three dimensional array of tiles.

This class also defines the type Region::type\_t which is the internal storage type for the tiles. Please note that the value 0 is used for empty tiles. region.hpp also defines three macros, REGION\_WIDTH, REGION\_HEIGHT and REGION\_DEPTH, which are used by the RegionPager objects to define the width, height and depth of the regions. Hopefully, these macros are temporary.

## RegionPager

The paging class is divided into two parts: RegionPagerBase and RegionPager. The abstract base class provides access to the correct region objects, via wrapper methods GetTile() and SetTile(). This class also allows the user to set the sizes of the regions, but please note that it is a Very Bad Idea to change this mid program.

The derived class, which takes two template parameters, overrides four abstract methods used for creating, and saving and loading the region objects. The derived class doesn’t do much itself, apart from bridging the gap between the base class and the functor classes provided as template parameters.

## Generator Functors

There are currently two generator functors, BlankGenerator and LuaGenerator. 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 either after creating it or before freeing it.

## 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.

## Lua’s Region API

The Region library is written to interact with Region objects as needed. The four most common functions are Save(r, saveDir), Load(r, saveDir), Create(r), and Unload(r). Each of these receives a Region object as a lightuserdata type, while Save() and Load() also receive strings containing the map’s save directory.

Accessors and mutators for the Region objects are also provided. The four functions mentioned above are usually called by the functor objects, and are defined as dummy functions by default. If you want to use them, I recommend redefining them in the server’s start up lua script.

The full list of available library functions (at the time of writing) is:

* Region.SetTile(r, x, y, l, v)
* Region.GetTile(r, x, y, l)
* Region.GetWidth(r)
* Region.GetHeight(r)
* Region.GetDepth(r)
* Region.GetX(r)
* Region.GetY(r)
* Region.Create(r)
* Region.Unload(r)
* Region.Load(r, saveDir)
* Region.Save(r, saveDir)

# Modding Support

TODO

# Communication Protocols

The primary method of communication is a custom UDP protocol.

TODO: NETWORKING DOCUMETNATION!!!

# 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 List

Clean up this document

Page breaks

Add more

*non-player characters? If I include a storyline further down the road, NPCs would be important.*

*factional warefare? PvP is probably the best focus for this game. Creating guilds, parties, etc. would be a good way to encourage communities on the same server. This might be a better fit for a single shard game.*

*at least a basic chat system is needed at some point*

*money & economy? again, something like this probably wouldn’t fit with a game aiming for 150 people per server.*

*mana nodes? If magic becomes a gameplay mechanic, mana nodes that increase that magic would be useful. Also, faction controlled mana nodes would be pretty cool. It seemed to work for Maridian 59.*

*world gates? Long distance teleportation. For root towns, it could be automatic, but maybe you need to have found other gate to be able to teleport there.*

*This needs some serious editing for consistency*

## Devving: First Name “Al”, Last Name “Gorithm”

I’m thinking about merging the design doc and the technical doc. They’re both dev docs, and I really don’t need two.

I’m trying to come up with an algorithm for combat, first by creating these stats:

HP Max

HP Now - 0 to HP Max

MP Max

MP Now - 0 to MP Max

Speed - “ticks” per second

Attack - physical attack

Defence - physical defence

Intelligence - Magical attack

Resistance - Magical defence

Accuracy

Evasion

Level

These would be stored as base stats, and then converted into derived stats by various means. Depending on your type of attack (physical or magical) the stats that are used at switched into the algorithm, otherwise the algorithm doesn’t change. This is similar to physical/special attacks in Pokemon.

For the active time battle system, each player needs 100 “ticks” in their meter 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.

The accuracy/evasion, which are percentages, are calculated like this:

return A.accuracy - A.accuracy \* D.evasion >= random(100)

So, for an attacker’s accuracy of 90, and a defender’s evasion of 20, there is a 72% chance of the attack landing. Fairly straight forward. I haven’t decided how to calculate critical hits. One possibility is if the RNG’s value is less than, say, 10% of the derived hit’s value (in this case, 7.2%), than the hit is critical. I could also add in a “luck” stat which would replace the 10% value, so that the user can up their own chances of landing a critical hit. Sorry if this wasn’t 100% clear, my documentation skills are not optimal.

A few nights ago, while barely awake, I wrote this down:

I know this isn’t very clear, and as shown by the corrections, I barely understood it. But the basic outline is this: subtract the defence from the attack until you reach half of the attack, then only reduce it by half of the defence. Continue this until you run out of the defence stat.

The problem is, I wrote this:

function f(a, d, m)

if a <= 1 then return 1 end

if a/2 <= d\*m then return a - d\*m end

return f(a/2, d\*m - a/2, m/2)

end

Apart from the fucked syntax, this algorithm is just plain broken. It’s amazing hte shit you write when you’re tired. Anyway, I’m currently loking for something that will work better. I’m wondering if the decreasing-impact curve above is good or not, so any feedback is appreciated.

Finally, I created a new release tag, release-0.2, which doesn’t have much to show in the front end, but a lot of trial-and-error work in the backend.

OK, any feedback is greatly appreciated. You can find the source here, and the most recent playable build here (windows only, sorry!).

## data dump & finished algorithm

(A.attack\*3 + A.attack\*(math.random(A.luck\*100)/10)) - (D.defence + D.defence\*D.evasion)

This is fairly stable, as far as the data dump goes. You can find the dump here. I made the minimum attack value 1 instead of 0, just because I feel that fits better. I could always add the attacker’s accuracy to their luck stat later, but whatever. Please note that 1: the player probably won’t be able to equip weapons with a higher level then them, and 2. the player won’t even be able to hit their enemy unless this passes:

random(100) <= A.accuracy - A.accuracy \* D.evasion