Tortuga Technical Document

By Kayne Ruse

Introduction 1

Languages and APIs 2

Gameplay Mechanics 2

Movement 2

Combat Portals 2

Combat 2

Permadeath 3

Player Character 3

Player Character Statistics 3

Items 3

Equipment 3

Mini Games 3

Server Mechanics 4

Server Structure 4

Data Storage 4

User Accounts 4

Scripting 4

Modding Support 4

Communication Protocols 4

Client Mechanics 4

Client Structure 4

Platforms 5

Game Controls 5

The Game Map 5

Tile 5

Region 5

Region Pager 6

Tile Sheet 6

Tile Sheet Manager 7

Map File Format 7

TODO List 7

# Introduction

This is the technical document for the Tortuga project. This is intended to plan out expected features of the game, as well as serve as an instructional introduction to modders and others interested in Tortuga’s development cycle.

# 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.

# Gameplay Mechanics

TODO

Walk around the world, combat

# Movement

The players can move around the world using pixel precision.

# Combat Portals

TODO

Multiplayer

server control

map design and generation

exploration

roguelike dungeons

control of server mechanics and scripts

travel between regions (world gates)

RPG mechanics like items, equipment, stats, etc.

Player Interactions

TODO

# Combat

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 party. Each player will be able to choose their own actions, including attacking with their equipped weapons, using spells or items, or fleeing the battle. 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.

The biggest innovation of this design is the drop-in/drop-out system. 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).

TODO

Player vs monsters

equations

# Permadeath

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

# Player Character

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:

Health - Life Remaining

Mana - Magic Remaining

Level - Skill Level

Attack - Offensive Ability

Defence - Defensive Ability

Strength

Speed

Luck

Magic Channelling - Magic Regen?

TODO

stats can be increased by methods other than levels and equipment

stats increased by items and levels

level progression

# Items

TODO

There will be many items in the server, whether they’re consumable items, equipment or other types.

# Equipment

TODO

# Mini Games

Although a series of mini games would be a nice addition to the game, they’re unnecessary, and a waste of resources while there are other more efficient uses of development time. Despite that, it should be possible to implement mini games further down the line.

# Server Mechanics

TODO

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

# Server Structure

<IMAGE LOST>

TODO

# Data Storage

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.

# Scripting

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

# Modding Support

TODO

# Communication Protocols

TODO

TCP vs UDP?

# Client Mechanics

TODO

Available options, how to connect to a server.

# Client Structure

TODO

# Platforms

At this stage, due to a limited scope and budget, this game will only be available on PC.

# Game Controls

This game will have both keyboard & mouse support, as well as generic controller support.

TODO

navigate through menus, move, select, etc.

# The Game Map

This section outlines the game’s map system. This system utilizes pagination to create a theoretically infinite game map, as well as supporting multiple tilesets in the same map. The goal of this design is to create a system with as much flexibility as possible, and simply enforcing a more rigid approach higher in the tool chain.

## Tile

The Tile class is the basic unit of the map system, and is explicitly a POD (plain old data) structure. A tile has these members:

X Position

Y Position

Depth

Width

Height

Tile Index

The tile’s X and Y positions are relative to their container region’s location. A tile’s depth allows multiple tiles to be drawn at the same location, and in the correct order; tiles with lower depths (including below zero) are drawn first. If a new tile has the same X position, Y position and depth as an existing tile, the old tile is overwritten.

The width and height members indicate the graphical size of the tile (not actually used when drawing), while the tile index is the specific tile for the sheet manager to draw. A negative value here is considered an error message.

## Region

The region class has these members:

X Position

Y Position

Width

Height

Tile Container

Each region in a certain map must have the same width and height, and it’s X and Y positions must be multiples of those width and height values, respectfully. The outcome of this restriction is a theoretically infinite grid of region objects.

Each region holds a set of tiles corresponding to the region’s bounds. The tiles’ X and Y positions are relative to the regions’, so moving the region will move the tiles as well. A region object is created or loaded when a tile is place in it’s bounds; similarly, if a region has no tiles it should be deleted or removed from memory.

The exact width and height of a region has no significant impact, other than loading or transmission costs. The width and height of a map can be adjusted as needed.

## Region Pager

The region pager class has these members:

Region Width

Region Height

On New Callback

On Delete Callback

Region Container

The Region Pager class holds a series of region objects, as well as creating and deleting them as needed. Every region theoretically exists at any time, so if a non-existent region object is requested, it is created and then returned. This class also has the Prune() method, which removes all regions out of bounds from memory, and the DrawTo method, which takes (among other things) the sheet manager for the map.

The width and height members must be set before the pager is used, and must not be changed while it still has regions loaded. These are used to create region objects as needed.

Each pager can also have two different callbacks set: “on new” and “on delete”. If either of these are set (that is, not null) then each region object’s address is passed to these after it is created or before it is destroyed, respectfully. The callbacks are intended to be used for domain specific processes, such as loading or saving data, or even requesting data from a remote server.

## Tile Sheet

A tileset is a series of tile graphics stored in a single file. The tile sheet class loads a tile set into memory, and provides utilities for drawing them to the screen. The tile sheet class has these members:

Image

X Count

Y Count

Total Count

Begin

End

The Image class is utilized heavily here by storing the graphical data and the tile size. Any file loaded into a sheet object must have all tile images arranged in a grid pattern, and they must all have the same width and height. The width and height must be provided when the file is loaded.

The X and Y counts are the number of tiles along the X and Y axis of the sheet’s image, and the total count is the number of tiles in the whole sheet (which is equal to the product of the X and Y counts).

Begin is the index of the first tile on the sheet (default is zero), and end is the index after the last tile (defaults to the value of total count). These indicate the range of the tiles, and are mostly used by the sheet manager. They are also used by the InRange() method, which checks to see if a certain tile is in that sheet.

## Tile Sheet Manager

This class has these members:

Tile Sheet Container

Range End

This class is a wrapper around a key-value container, using strings as the keys. Given a specific tile index, this class will draw the correct tile from the loaded sheets, or it throws an error.

Also, this class keeps track of the end of the sheet’s ranges.

TODO

# Map File Format

TODO

# TODO List

Clean up this document

Page breaks

Separate the deprecated map doc

Add more