Game Design Document Extended.

This purpose of this document is to further explain the state changes within the game.

Much of the work I did was on animations. I use various states to determine what animations to play and at what times.

The state of the player, and any sprite for that matter, can change in a number of ways. In terms of movement/animation (the two are intertwined in my project) the player can be facing in any one of 8 different directions to move. Any sprite can be taking one of any 6 actions. Finally, a sprite can either be in battle or not.

These states are combined to define what type of animation is played.

Whenever the player levels up, the attack and health stats are affected. Whenever a player wins a battle, the experience and gold stats are increased.

Now Ill go a bit more in depth to the various systems that are in my game.

Scene Class:

The scene class encapsulates the window object that is used to display all the sprites.

// This function adds a sprite that will be used as the background of the scene

void AddBackground(MySprite\* background);

// unused

void AddHud(MySprite\* HUD);

// updates each sprite, will handle out of bounds behavior

void UpdateSprites();

// draw all the sprites in the sprite list to the 1st buffer

void DrawSprites();

// displays the frame and also

// will wait until the entire frame has finished : this is based on FPS

void DisplayFrame();

// add sprite to the list of sprites to be drawn each cycle

void AddSprite(ReinerSprite\* new\_sprite);

// checks the x and y of a sprite and returns true if its in the screen and false if outside

bool CheckBounds(MySprite\* sprite\_to\_check);

// utility function so that i can check only the players bounds

bool CheckPlayerBounds(ReinerSprite\* player);

// goes through the sprite list and checks if any collisions occur

// returns the sprite that collides with the player

ReinerSprite\* CheckCollisions();

// Bounces the sprite off the edge of the screen

void Bounce(MySprite\* sprite\_to\_bounce);

// teleports the sprite to the opposite side of the scree

void Teleport(MySprite\* sprite\_to\_tp);

// reverse the direction of the sprite to the opposite direction

void ReverseDirection(ReinerSprite\* sprite\_to\_reverse);

// getters for height and width

int GetHeight();

int GetWidth();

The Reiner Sprite is the other major structure in this project. It derives from the sprite class I created off sfml’s sprite. This is a hefty object that stores much of the data to be used throughout the program. Every entity in the game is based off ReinerSprite, even the player.

// updates the sprites position based on its speed and direction

void UpdateSprite();

// sets the sprite sheet to be used by the sprite

void SetSpriteSheet(std::string file\_name);

// function that animates the sprite based on its current state. Will return true if the end of an anim cycle is reach and false otherwise

bool Animate();

// set a direction or action state

void SetNewDir(int new\_dir);

virtual void SetNewAction(int new\_action);

// getters for direction and action state

int GetDirState();

int GetActionState();

// get or set the event state : Two events are peace and battle

void SetEventState(int action\_state);

int GetEventState();

// getters and setters for stats

void SetAttack(int attack);

int GetAttack();

void SetHealth(int health);

int GetHealth();

void SetMaxHealth(int max\_health);

int GetMaxHealth();

// will return true if player levels up, false otherwise

bool SetExp(int exp);

int GetExp();

// sets level and also defines behavior for when a sprite levels up

void SetLevel(int level);

int GetLevel();

// attack function will take a sprite to attack

// returns true if it kills the sprite, otherwise false

bool Attack(ReinerSprite\* attacked\_sprite);

// adds an item object to the sprites inventory

void AddItemToInventory(ItemSprite\* new\_item);

std::vector<ItemSprite\*> GetInventory();

// gets the back drop to the inventory panel

sf::RectangleShape GetBackdrop() {

return inventory\_backdrop;

}

// sets and gets whether inventory should be shown or not

void SetShowInventory(bool choice);

bool GetShowInventory();

// set the sprites gold amount

void SetGold(int some\_gold);

int GetGold();

// set the exp reward if the sprites is killed

int GetExpReward();

int GetGoldReward();