**Sprite(x,y,width,height,container)**

GetCenterX: returns the x position of the center point on the Sprite

GetCenterY: returns the y position of the center point on the Sprite

getCenter: returns the center point of the Sprite

getActualX: gets the actual x that was passed in

getActualY: gets the actual y that was passed in

getX: gets the percentage x based on the container

getY: gets the percentage y based on the container

getWidth: returns the width as a percentage based on the container

getHeight: returns the height as a percentage based on the container

setX(x): sets the actual x and recalculates the percentage x

setY(y): sets the actual y and recalculates the percentage y

setWidth(w): sets the actual width and recalculates the percentage width

setHeight(h): sets the actual height and recalculates the percentage height

setLoc(x,y): sets the actual x and actual y and recalculates the percentages

moveX(dx): adds onto the current percentage x and sets the spdX and spdY

moveY(dy): adds onto the current percentage y and sets the spdY and spdX

getXSpd: returns the current spdX

getYSpd: returns the current spdY

contains(x,y): returns a Boolean value depending on if x and y are positioned on top of the Sprite

intersects(sprite): returns a Boolean value depending on if the sprite is intersecting with the current Sprite

resize: recalculates the percentage x, y, width, and height of the Sprite based on the container

draw(graphics): draws the current sprite

**ImageSprite(x,y,w,h,image,container)**

setImage(img): sets the current image to the new img

getImage: returns the current image

draw(graphics): draws the current image

**AnimatedSprite(x,y,w,h,container,img,frameCount,fps)**

Play(loopCount): tells the animation to play and how many times it should play (-1 is infinite)

setReverse(Boolean): plays the animation backwards if true

update(deltaTime): changes the current frame of the animation depending of deltaTime, fps, and frameCount

pause(Boolean): pauses the animation if false

isPlaying: returns true if the animation is running

goToFrame(frame): sets the current frame to the inputted frame number

**Player(x,y,w,h,container,img,frameCount,fps,stats,health,mana,username)**

canMove: returns true if the player can move

setMove(Boolean): the player cannot move if set to false

getInv: returns the current inventory of the player

getStats: returns the current stats of the player in the form of an array of size 4

getUser: returns the username

getCurrentHealth: returns the players current health

getMaxHealth: returns the players maximum health

getLevel: returns the players level

getTotalDamage: returns the players strength level plus the damage of the current weapon equipped

getTotalSpd: returns the players agility level plus the speed of the current weapon equipped

getCurrentMana: returns the players current mana

getMaxMana: returns the players maximum mana

getSpells: returns the players current spells in the form of an array

addSpell(spell): adds a new spell to the players current spells array

getCurrentWeapon: returns the players equipped weapon

getCurrentArmor: returnst he players equipped armor

addItem(item): adds a new item to the players current inventory

removeItem(itemName): removes the item from the players inventory based on name

gainXP(xp): adds the xp gained to the players totalXP

getMaxExp: returns the remaining xp needed to achieve the next level

takeDamage(dmg): subtracts the damage dealt to the player and checks to see if the player is still alive

heal(hp): adds the amount healed to the players currentHP

useMana(mp): subtracts the used mana from the players currentMP

gainMana(mp): adds the gained mana to the players currentMP

levelUp: adds 50 to max hp, 10 to max mp, and increases the 3 main stats depending on the total level of the player

isAlive: returns true if the player is still alive

equip(item): Equips the selected item if it is a weapon or armor piece

unEquip(type): removes the players currentWeapon or currentArmor and places it into the players inventory

update: checks to see if the player has reached the next level and runs the animations/movments

**Enemy(x,y,w,h,container,img,frameCount,fps,stats,health,mana)**

getCurrentHealth: returns the enemy current health

getMaxHealth: returns the enemy maximum health

heal(hp): adds the amount healed to the enemy currentHP

useMana(mp): subtracts the used mana from the enemy currentMP

takeDamage(dmg): subtracts the damage dealt to the player and checks to see if the enemy is still alive

getCurrentMana: returns the enemy’s current mana

getMaxMana: returns the enemy’s maximum mana

getStats: returns the current stats of the enemy in the form of an array of size 4

getSpells: returns the enemy’s current spells in the form of an array

**Weapon(x,y,w,h,img,container,name,spd,damage,price)**

getSpd: returns the speed of the weapon

getDamage: returns the amount of damage dealt by the weapon

getPrice: returns the price of the weapon

getName: returns the name of the weapon

getType: returns the type weapon as a string

setName(name): sets the current name to the specified name

**Armor(x,y,w,h,img,container,name,defence,price)**

getDefence: returns the defense of the armor

getPrice: returns the price of the armor

getName: returns the current name of the armor

getType: returns the type armor as a string

setName(name): sets the current name to the specified name

**QuestObject(x,y,w,h,img,container,name,price,sellable)**

getPrice: returns the price of the object

getName: returns the name of the object

getType: returns the type Object as a string

setName(name): sets the current name to the specified name

isSellable: returns true if it is sellable

**Spell(x,y,w,h,img,container,name,damage,manaCost)**

getName: returns the name of the spell

getType: returns the type Spell as a string

setName(name): sets the current name to the specified name

getManaCost: returns the mana cost of the spell

getDamage: returns the total damage of the spell

**Tile(x,y,w,h,image,container,id)**

getID: returns the id of the tile

setID(id): sets the current id to the specified id

**DialogueBox(x,y,w,h,container,message)**

setMessage(msg): sets the current message to the specified message

draw(graphics): draws the message to the container, if the length of the message is bigger than the container the message will be split into multiple lines and then displayed

**StateMachine()**

update(deltaTime,mX,mY): updates the current state on the stack

draw(graphics): draws the current state on the stack

addState(state): adds a new state to the StateMachine’s usable states

changeState(stateIndex,enemy): pushes the specified state onto the stack and passes in an optional enemy onto the new state

revertState: pops the current state of the stack and runs its onExit function

returnToMenu: pops all of the current states off the stack except for the first state on the stack which will be the menu state

**MainMenuState(canvas,stateMachine,mainCharacter)**

onEnter: sets up the two buttons that are present on the main menu

draw(graphics): draws the buttons and the title

update(deltaTime,mX,mY): checks to see if any of the buttons are pressed and then runs their functions

**LocalGameState(canvas,stateMachine,mainCharacter)**

onEnter: draws the tile map onto the canvas along with the player

onExit: removes all eventListeners, the tile map, and the character

draw(graphics): draws the viewport along with the maincharacter and tiles

update(deltaTime,mX,mY): checks for collision between the player and collidable objects

**Inventory State(canvas,stateMachine,mainCharacter)**

onEnter: sets up the needed buttons along with the mainCharacters current items

onExit: removes all buttons and items from canvas

updateInv: checks if new items were added or if old items were removed from the players current inventory and updates the images on the canvas

update(deltaTime,mX,mY): checks to see if the player pressed a specified button and runs its functions

draw(graphics): draws all of the buttons and the players inventory

**StatsState(canvas,stateMachine,mainCharacter)**

onEnter: sets up all requited buttons, decorations, and variables needed for displaying the mainCharacters current stats

onExit: removes all buttons and decorations from the canvas

update(deltaTime,mX,mY): checks to see if any of the buttons were clicked and if so then runs their functions

draw(graphics): draws all the current buttons and decorations to the canvas

**MapSystem(x,y,container)**

generateMap(type,layered): if layered is false, the current map is selected using type and then each integer in the array is set to a specific tile based on the integers value. These are then pushed onto the currentMap array. If layered is true, then each layer is looped over one by one, being added into currentMap as an array of arrays.

Draw(graphics) draws the currentMap to the screen, starting at layer 0.

getCollision: returns all collidable objects that are currently drawn on the map

getMapDetails: returns the column size, row size, player position X, player position Y and the total length of the map

changeMap(type,layered): changes the current map to a new generatedMap

getCurrentMap: returns the current map being drawn

**battleState(canvas,stateMachine,mainCharacter)**

onEnter(enemy): sets up the battle stances of the mainCharacter and current enemy, the hp and mp bars of both characters and then adds an action depending on whether the player is faster or the enemy is faster

update(deltaTime,mX,mY): updates the current battle state and updates the width of the hp bars for both characters

draw(graphics): draws both hp bars along with the battle stances and the current battle state

**BattleTick(battleState,stateMachine,mainCharacter,enemy,canvas,actions)**

update(deltaTime,mX,mY): checks to see if the player/enemy has chosen an action to perform and then changes the state depending on the action

draw(graphics): draws the current actions present

addAction(action): Adds a new action to the actions array

**PAction(mainCharacter,enemy,canvas)**

getReady: returns a true if the action is ready to run

getType: returns type player as a string

getSpd: returns the speed of the action

getPlayer: returns the player associated with this action

getEnemy: returns the enemy associated with this action

getActionType: returns the type of action the player has chosen

updateSpells: updates the list of spells depending on the characters list of spells

update(deltaTime,mX,mY): checks to see if the player chooses an action and then runs that actions functions

draw(graphics): draws the buttons that allows the player to choose an action

**EAction(mainCharacter,enemy,canvas)**

getReady: returns a true if the action is ready to run

getType: returns type player as a string

getSpd: returns the speed of the action

getPlayer: returns the player associated with this action

getEnemy: returns the enemy associated with this action

getActionType: returns the type of action the enemy has chosen

updateSpells: updates the list of spells depending on the enemy’s list of spells

update(deltaTime,mX,mY): runs the AI program and chooses an action for the enemy

**BattleAction(stateMachine,battleState,canvas)**

onEnter(action): sets up the variables depending on the action being executed

onExit: resets the current action to 0 and current spells

update: checks to see if the action is either enemy or the player and then runs the action against the opposing character

draw(graphics): draws the animation for the current action

**Node(leftChild,rightChild,parent,isRoot)**

getLeftChild: returns left child

getRightChild: returns right child

getParent: returns parent

isRoot: returns true if the node is a root node

setLeftChild(node): sets the current left child to the specified node

setRightChild(node): sets the current right child to the specified node

**Clamp(value,min,max)**

Keeps the specified value between the max and the min values

**collisionDirection(mainCharacter,tile)**

keeps the player from moving past unpassable objects depending on which direction the character was moving in when the collision occurred

**mergeSort(actions)**

Sorts the specified array from smallest value to largest value depending on the speeds of the actions

**merge(left,right)**

merges the two arrays together depending on which value of the array is smaller