Mori Engine Documentation

Jennifer Chen, Chris Shia, Rit Gangopadhyay, Chris Shia, Maryeva Gonzalez

Summary:

Tilemap:

- *initialize()*
- *update()*
- *render()*
- *isSolid(int pixelX, int pixelY)*
- *getTileIndex(int pixelX, int pixelY)*
- *playerOnWinTile(GameObject player)*
- *paintTile(index, tileState, app)*
- *hasEnemy(int index)*
- *setEnemy(int index)*
- *hasCoin(int index)*
- *setCoin(int index)*
- *isWinTile(int index)*

Player:

- *setTilemap(Tilemap tm)*
- *Input(GameObject player)*
- *Update(GameObject player)*
- *resetToSpawn(GameObject player)*

CollisionUtility:

- *checkAABB(SDL_FRect a, SDL_FRect b)*
- *gameObjectsOverlap(GameObject a, GameObject b)*

LevelEditorScene:

- *resetLevel()*
- *forceBuildMode()*
- *forcePlayMode()*
- *hasPlayer()*
- *load()*
- *input(SDL_Event* event)*
- *update(float dt)*
- *render()*
- *spawnStarOnTile(int tileIndex)*
- *spawnCoinOnTile(int tileIndex)*
- *spawnPlayerOnTile(int tileIndex)*
- *SpawnEnemyonTile(int tilex, int tiley, int tileh, int tilew, int tileIndex)*

GameObject:

- *this(SDL_Renderer* renderer, string bitmapFilePath, string type)*
- *Update()*
- *Render(SDL_Renderer* renderer)*
- *SetSheetRectangle(int x)*
- *SetPosition(int x, int y)*

SetSpritePosition(int x, int y)
SetDimensions(int w, int h)
SetVelocity(int vx, int vy)
SetSpeed(int speed)

Enemy:

CreateStillEnemyFactory(SDL_Renderer renderer)*
CreateUDEnemyFactory(SDL_Renderer renderer)*
CreateLREnemyFactory(SDL_Renderer renderer)*

Animation:

setAnimation(GameObject obj, string stateName)
void AdvanceFrame(GameObject obj)
void ApplyFrameToSheetRect(GameObject obj)

Resources:

ResourceManager GetInstance(string configPath = null)
LoadConfig(string configPath)
ParseSpriteData()
LoadTexture(SDL_Renderer renderer, string key)*
LoadEnemyTexture(SDL_Renderer renderer, string key)*
LoadStarTexture(SDL_Renderer renderer, string key)*

Level Scene Manager:

this()
void initializeLevel(int index, LevelEditorScene scene)
private void handleWinCondition(int currentIndex, bool forceBuild)
LevelEditorScene getCurrentLevel()
void switchToLevel(int newIndex)
void switchToLevelAfterWin(int newIndex, bool forceBuild)
void input(SDL_Event* event)
void update(float dt)
void render()

Tilemap:

initialize

```
void initialize()
```

Initializes the tilemap's internal arrays.

Details:

Creates and also sizes the internal arrays that are used to store the tile GameObjects, tile types, enemy occupancy flags, and coin flags.

Parameters:

None.

Returns:

Nothing.

update

```
void update()
```

Updates all the tiles in the tilemap.

Details:

Iterates through all the tile GameObjects and calls their update() methods.

Parameters:

None.

Returns:

Nothing.

render

```
void render()
```

Renders all tiles to the screen

Details:

Iterates through all the tile GameObjects and calls their render() methods.

Parameters:

None.

Returns:

Nothing.

isSolid

```
bool isSolid(int pixelX, int pixelY)
```

Determines whether the tile at a pixel coordinate is solid.

Details:

Converts the pixel position provided into a tile index, when retrieves the tileType and determines whether the tile is a wall (tileType 2).

If a pixel is ount of bounds, the function returns false.

Parameters:

int pixelX	pixel X coordinate
int pixelY	pixel Y coordinate

Returns:

true if the tile is solid, otherwise false.

getTileIndex

```
int getTileIndex(int pixelX, int pixelY)
```

Returns the tile index for a pixel position.

Details:

Converts the world-space pixel coordinate into a tile index. If a pixel coordinate lies outside the map boundaries, the function returns -1.

Parameters:

int pixelX	pixel X coordinate
int pixelY	pixel Y coordinate

Returns:

true if the tile is solid, otherwise false.

playerOnWinTile

```
bool playerOnWinTile(GameObject player)
```

Checks whether the player is currently standing on the win tile.

Details:

Converts the player's center pixel position into a tile index. It then checks if the tileType is 3.

Parameters:

GameObject player	the Player game object

Returns:

true if the player is on the win tile, otherwise false.

paintTile

```
void paintTile(int index, int tileState, GameApplication* app)
```

Changes the tile at a certain index to a different tileType.

Details:

Updates the tile's spritesheet frame, collision type, and win, coin, and enemy flags based on the selected tileState in the editor.

Parameters:

int index	Tile index
int tileState	Editor-selected tile type
GameApplication* app	Reference to the GameApplication

Returns:

Nothing.

hasEnemy

```
bool hasEnemy(int index)
```

Returns whether an enemy is registered on the tile.

Details:

Returns the value stored in the enemy occupancy array.

Parameters:

int index	the tile index

Returns:

true if an enemy occupies this tile, otherwise false.

setEnemy

```
void setEnemy(int index, bool value)
```

Marks whether a tile contains an enemy.

Details:

Marks whether a tile contains an enemy.

Note: Changing this value does not create or destroy actual enemy GameObjects, only the occupancy metadata.

Parameters:

int index	Tile index
bool value	Whether an enemy occupies this tile

Returns:

Nothing.

hasCoin

```
bool hasCoin(int index)
```

Returns whether a tile contains a coin.

Details:

Checks the coin occupancy array.

Parameters:

int index	Tile index

Returns:

true if a coin occupies this tile, false otherwise.

setCoin

```
void setCoin(int index, bool value)
```

Sets whether a tile contains a coin item.

Details:

Marks whether a tile contains a coin.

Note: Changing this value does not create or destroy actual coin GameObjects, only the occupancy metadata.

Parameters:

int index	Tile index
bool value	Whether a coin occupies this tile

Returns:

Nothing.

isWinTile

```
bool isWinTile(int index)
```

Determines whether the tile at a pixel coordinate is solid.

Details:

Converts the pixel position provided into a tile index, when retrieves the tileType and determines whether the tile is a wall (tileType 2).

If a pixel is ount of bounds, the function returns false.

Parameters:

int pixelX	pixel X value
int pixelY	pixel Y value

Returns:

true if the tile is solid, otherwise false.

Player:

setTileMap

```
void setTilemap(Tilemap tm)
```

Assigns the tilemap that the player interacts with.

Details:

This function sets a reference to teh active Tilemap for collision checks, solid tile and win tile checks for the layer.

Note: This should be called before any update logic or the player will be unable to collide with the environment.

Parameters:

Tilemap tilemap	The Tilemap that the player moves on

Returns:

Nothing

Update

```
void Update(GameObject player)
```

Update function for the player game object.

Details:

This function updates all core functionality for the PlayerScript. This includes reading input, movement, collision, enemy interaction, win tile detection, and also animation handling.

Parameters:

GameObject player	The GameObject instance that owns this script.

Returns:

Nothing.

resetToSpawn

```
void resetToSpawn(GameObject player)
```

Resets the player's position to the stored spawn location.

Details:

Moves the player back to the spawn tile that was previously stored when the player object was created. When the player dies, hits an enemy, or starts a level the player will spawn here.

Parameters:

GameObject player	The player GameObject whose position is being reset.

Returns:

true if the tile is solid, otherwise false.

CollisionUtility:

checkAABB

```
static bool checkAABB(SDL_FRect a, SDL_FRect b)
```

Performs axis-aligned bounding box.

Details:

This function essentially performs an axis-aligned bounding box collision detection on two SDL_FRect objects.

Parameters:

SDL_FRect a	First rectangle
SDL_FRect b	Second rectangle

Returns:

true if the rectangles overlap, otherwise false.

gameObjectsOverlap

```
static bool gameObjectsOverlap(GameObject a, GameObject b)
```

Checks whether two GameObjects overlap based on their transform rectangles.

Details:

Moves the player back to the spawn tile that was previously stored when the player object was created. When the player dies, hits an enemy, or starts a level the player will spawn here.

Parameters:

SDL_FRect a	First rectangle
SDL_FRect b	Second rectangle

Returns:

true is the bounding rectangles of both objects overlap, false otherwise.

LevelEditorScene:

resetLevel

```
void resetLevel()
```

Resets the level to its initial state.

Details:

This function restores all the changes objects in the scene to their initial conditions.

Parameters: None.

Returns:

Nothing

forceBuildMode()

```
void forceBuildMode()
```

Switches the current scene into editor mode.

Details:

This function forces the LevelEditorScene out to play mode and into the level editor. Used when user presses editor button or when level must return to a safe editable state.

Parameters:

None.

Returns:

Nothing.

forcePlayMode()

```
void forcePlayMode()
```

Switches the current scene into player mode.

Details:

This initiates gameplay. Triggered when the user clicks the play button.

Parameters:

None.

Returns:

Nothing.

hasPlayer()

```
void hasPlayer()
```

Checks whether a player object exists in the scene.

Details:

This is a check to see if the GameObject has been spawned via the editor tools.

Parameters:

None.

Returns:

true if a player GameObject exists, otherwise false.

load()

```
override void load()
```

Loads the scene when it becomes the active scene.

Details:

This method is called by the SceneManager.

Parameters:

None.

Returns:

Nothing

input

```
override void input(SDL_Event* event)
```

Handles all mouse and keyboard input in both editor and play modes.

Details:

None.

Parameters:

SDL_Event* event	the SDL event received

Returns:

Nothing.

update

```
override void update(float dt)
```

Updates the tilemap, objects, UI, and gameplay logic.

Details:

None.

Parameters:

float dt	The time elapsed since last frame

Returns:

Nothing.

render

```
override void render()
```

Renders the tilemap, objects, UI for the level editor scene.

Details:

While in Build mode, this also renders selection highlights. In play mode renders only world objects.

Parameters:

None

Returns:

Nothing.

spawnStarOnTile

```
GameObject spawnStarOnTile(int tileIndex)
```

Creates a star collectible on the specified tile.

Details:

None.

Parameters:

int tileIndex	The tile index to place the star

Returns:

The created star GameObject.

spawnCoinOnTile

```
GameObject spawnCoinOnTile(int tileIndex)
```

Creates a coin collectible on the specified tile.

Details:

None.

Parameters:

int tileIndex	The tile index to place the coin

Returns:

The created coin GameObject.

spawnPlayerOnTile

```
GameObject spawnPlayerOnTile(int tileIndex)
```

Places the player at the specified file.

Details:

Note: This function is required before entering play mode.

Parameters:

int tileIndex	The tile index to place the player

Returns:

The created player GameObject.

void SpawnEnemyonTile(int tilex, int tiley, int tileh, int tilew, int tileIndex)

spawnEnemyOnTile

```
void SpawnEnemyOnTile(int tilex, int tiley, int tileh, int tilew, int tileIndex)
```

Places the enemy at the specified file.

Details:

Note: This function is required before entering play mode.

Parameters:

int tileIndex	The tile index to place the enemy

Returns:

The created enemy GameObject.

spawnEnemyOnTile

`void SpawnEnemyOnTile(int tilex, int tiley, int tileh, int tilew, int tileIndex)`

Places the enemy at the specified file.

Details:

Note: This function is required before entering play mode.

Parameters:

int tileIndex	The tile index to place the enemy

Returns:

The created enemy GameObject.

GameObject:

this

`this(SDL_Renderer* renderer, string bitmapFilePath, string type)`

Constructs a new GameObject with the given texture and name.

Details:

Note: all objects in the game originate here.

Parameters:

SDL_Renderer* renderer	SDL renderer used for drawing
string bitmapFilePath	The initial texture of the object
string type	A string identifier

Returns:

A constructed GameObject instance.

Update

`void Update()`

Updates all attached components and the attached script.

Details:

None.

Parameters:

float dt	The time elapsed since last frame.

Returns:

Nothing.

Render

`void Render(SDL_Renderer* renderer)`

Renders the GameObject using its TransformComponent and current texture.

Details:

If the object has animation, the animation frame rectangle overrides the default texture rectangle.

Parameters:

None.

Returns:

Nothing.

SetPosition

```
void SetPosition(int x, int y)
```

Sets the world-space position of the GameObject.

Details:

Accesses the TransformComponent and updates its rectangle's x and y coordinates..

Parameters:

int x	X Pixel coordinates
int y	Y Pixel coordinates

Returns:

Nothing.

SetSpritePosition

```
void SetSpritePosition(int x, int y)
```

Sets the sprite position of the GameObject

Details:

None.

Parameters:

int x	X Pixel coordinates
int y	Y Pixel coordinates

Returns:

Nothing.

SetDimensions

```
void SetDimensions(int w, int h)
```

Resizes the GameObject’s bounding rectangle.

Details:

Updates the TransformComponent to apply a new width and height.

Parameters:

int x	Width in pixels
int h	Height in pixels

Returns:

Nothing.

Enemy

CreateStillEnemy

```
Enemy CreateStillEnemyFactory(SDL_Renderer* renderer)
```

Creates an enemy that remains still.

Details:

This function constructs an enemy GameObject.

Parameters:

SDL_Renderer* renderer	Active SDL renderer

Returns:

A fully constructed still enemy GameObject.

CreateUDEnemyFactory

```
Enemy CreateUDEnemyFactory(SDL_Renderer* renderer)
```

Creates an enemy that moves vertically.

Details:

This function constructs an enemy GameObject.

Parameters:

SDL_Renderer* renderer	Active SDL renderer

Returns:

A fully constructed up-down enemy GameObject.

CreateLREnemy

```
Enemy CreateLREnemyFactory(SDL_Renderer* renderer)
```

Creates an enemy that moves horizontally.

Details:

This function constructs an enemy GameObject.

Parameters:

SDL_Renderer* renderer	Active SDL renderer

Returns:

A fully constructed right-left enemy GameObject.

Animation:

setAnimation

```
void setAnimation(GameObject obj, string stateName)
```

Switches the GameObject's animation to a named animation state.

Details:

None.

Parameters:

GameObject obj	GameObject whose animation is being modified
string stateName	String identifier of the animation

Returns:

Nothing.

AdvanceFrame

```
static void AdvanceFrame(GameObject obj)
```

Advances the GameObject's active animation based on how much time has passed.

Details:

None.

Parameters:

GameObject obj	GameObject whose animation is being updated

Returns:

Nothing.

ApplyFrameToSheetRect

```
static void ApplyFrameToSheetRect(GameObject obj)
```

Updates the GameObject's render source rectangle to match the current animation frame.

Details:

None.

Parameters:

GameObject obj	GameObject whose animation is being updated

Returns:

Nothing.

Resources:

GetInstance

```
static ResourceManager GetInstance(string configPath = null)
```

Returns the global ResourceManager instance, creating it if it does not yet exist.

Details:

None.

Parameters:

string configPath	Path to the engine JSON configuration

Returns:

A singleton ResourceManager.

LoadConfig

```
void LoadConfig(string configPath)
```

Loads the JSON configuration file.

Details:

None.

Parameters:

None.

Returns:

Nothing.

ParseSpriteData

```
void ParseSpriteData()
```

Parses data in the format of SpritaeState

Details:

None.

Parameters:

None.

Returns:

Nothing.

LoadTexture

```
SDL_Texture* LoadTexture(SDL_Renderer* renderer, string key)
```

Loads the main spritesheet defined in config.

Details:

None.

Parameters:

SDL_Renderer* renderer	SDL renderer to create textures
string key	Dictionary key used to store the texture

Returns:

An SDL_Texture pointer

Level Scene Manager:

this()

this

```
this()
```

Initializes LevelManager with a fixed-size array of 3 levels.

Details:

None.

Parameters:

None.

Returns:

A new LevelManager instance.

initializeLevel

```
void initializeLevel(int index, LevelEditorScene scene)
```

Registers a LevelEditorScene into the level array.

Details:

None.

Parameters:

int index	Slot in the level array (0-2)
LevelEditorScene	The LevelEditor Scene instance to install

Returns:

Nothing

handleWinCondition

```
private void handleWinCondition(int currentIndex, bool forceBuild)
```

Processes a win in the active level and switches to the next level if available

Details:

None.

Parameters:

int currIndex	Index of th elevel that was just completed
forceBuild	Whether to force entering build mode

Returns:

Nothing.

getCurrentLevel

```
LevelEditorScene getCurrentLevel()
```

Returns the currently active LevelEditorScene.

Details:

None.

Parameters:

None.

Returns:

The active LevelEditorScene or null if this has not been assigned.

switchToLevel

```
void switchToLevel(int newIndex)
```

Switches the engine to a different level.

Details:

None.

Parameters:

int newIndex	The index of the level to switch with.

Returns:

Nothing.

switchToLevelAfterWin

```
void switchToLevelAfterWin(int newIndex, bool forceBuild)
```

Switches level after a win, optionally entering play mode automatically

Details:

None.

Parameters:

int newIndex	The index of the level to switch with.
bool forceBuild	Whether to enter build mode instead of play mode.

Returns:

Nothing.