Tutorial 11 Enemy AI

Before starting this tutorial, you must complete the last one and have it working. Failure to do so will result in not being able to complete these steps. It is your choice but is recommended to crate a new branch each tutorial.

We are now coming to final steps in these tutorials, here we will be creating the enemy characters based off the original Character implementation using inheritance. You should have already created a header and source file for Character in a previous tutorial. You will now inherit in the same manner as you have done with the CharacterMario class.

Firstly, add the new images (Koopa and coins), on Blackboard to the Images folder of your assignment.

Now you have this we can concentrate on coding out enemy behaviour. Add a CharacterKoopa class to your solution.

Enemy characters in Mario Bros require some additional variables and functions not found in Character. To start with the Koopa character texture has 2 sprites on it. This is a sprite sheet. A simple one, but a sprite sheet, nonetheless. For us to be able to chop out the image we require when rendering we need to add the following private variables, firstly though, ensure you have set up your file to inherit from Character then add the variables:

```
float m_single_sprite_w;
float m_single_sprite_h;
```

After these, add two more private variables; these will be used to enable the enemy to be flipped and damaged for a period of time.

- A Boolean called m_injured.
- And another float called m_injured_time.

Next, create two public functions along with the constructor, destructor:

```
void TakeDamage();
void Jump();
```

Then, add a private function called FlipRightwayUp, this should be a void function.

Finally, the public override functions for Render and Update (if unsure how to do this check your CharacterMario files).

Moving across to the CharacterKoopa.cpp file you should implement the constructor and have it set up all the variables of the Character base class and its own variables. The following code snippet shows how I have implemented this. Notice that I have included additional parameters. You can include these in yours too, some may require moving or creating.

```
CharacterKoopa::CharacterKoopa(SDL_Renderer* renderer, string imagePath, LevelMap* map,
Vector2D start_position, FACING start_facing, float movement_speed) : Character(renderer,
imagePath, start_position, map )
{
	m_facing_direction = start_facing;
	m_movement_speed = movement_speed;
	m_position = start_position;
	m_injured = false;

	m_single_sprite_w = m_texture->GetWidth() / 2;
	m_single_sprite_h = m_texture->GetHeight();
}
```

1. The first function we are going to look at is the TakeDamage() function. Within this function do the following:
    a. Set m_injured to true.
    b. Set m_injured_time to equal the amount of time you want Koopa to be injured for. I have created a constant variable called INJURED_TIME and set it to 2.5f. This should be placed in Constants.h
    c. Call the Jump function.
2. Next, the Jump function should have the following, it is similar to what has been implemented with Mario:

    ```
    if(!m_jumping)
    {
    	m_jump_force = INITIAL_JUMP_FORCE_SMALL;
    	m_jumping = true;
    	m_can_jump = false;
    }
    ```

3. Now the FlipRightwayUp function. This is simply a matter of changing m_facing_direction to face the opposite direction, setting m_injured to false and calling the Jump function.

We now only have two functions remaining to flesh out. These are the two that will override the behaviour in the base class. Let's first take a look at the Render function. The comments explain the behaviour taking place, so make sure you read them and understand.

```
//variable to hold the left position of the sprite we want to draw
int left = 0.0f;

//if injured move the left position to be the left position of the second image of the
sprite sheet
if (m_injured)
       left = m_single_sprite_w;


//get the portion of the sprite sheet you want to draw
//                                                  {xPos, yPos, width of sprite,
height of sprite}
SDL_Rect portion_of_sprite = { left,0,m_single_sprite_w, m_single_sprite_h };

//determine where you want it drawn
SDL_Rect destRect = { (int)(m_position.x), (int)(m_position.y), m_single_sprite_w,
m_single_sprite_h };

//then draw it facing the correct direction
if (m_facing_direction == FACING_RIGHT)
{
       m_texture->Render(portion_of_sprite, destRect, SDL_FLIP_NONE);
}
else
{
       m_texture->Render(portion_of_sprite, destRect, SDL_FLIP_HORIZONTAL);
}
```

Finally, the Update method. The majority of a character's update takes place in Character, we are calling this and then extending the behaviour specific to Koopa.

```
//use the code within the base class
Character::Update(deltaTime, e);

if (!m_injured)
{
	//enemy is not injured so move
	if (m_facing_direction == FACING_LEFT)
	{
		m_moving_left = true;
		m_moving_right = false;
	}
	else if (m_facing_direction == FACING_RIGHT)
	{
		m_moving_right = true;
		m_moving_left = false;
	}
}
else
{
	//we should not be moving when injured
	m_moving_right = false;
	m_moving_left = false;

	//count down the injured time
	m_injured_time -= deltaTime;

	if (m_injured_time <= 0.0)
		FlipRightwayUp();

}
```

Now that we have the enemy class in place, we can start to use it. In GameScreenLevel1.h, define two new private methods as shown below:

```
void UpdateEnemies(float deltaTime, SDL_Event e);
void CreateKoopa(Vector2D position, FACING direction, float speed);
```

1. Rather than using an array we are going to be using a vector to hold the enemy characters, the benefits of using a vector over an array is that vectors are dynamic. Add the following private variable declaration (you will need to include the <vector> header file to use them):
   ```
   vector<CharacterKoopa*> m_enemies;
   ```
2. In the GameScreenLevel1.cpp file, in the destructor, clear your vector like so. This ensures that any objects with in it are disposed of.
   ```
   m_enemies.clear();
   ```
3. Then, using this new vector of enemies we can render every enemy in the list. This allows us to have multiple enemies all being drawn to the screen. You should be rendering your

enemies in your GameScreenLevel1 Render method and it should take place before your Mario and Background textures are rendered.

```
//draw the enemies
for(int i =0; i < m_enemies.size(); i++)
{
	m_enemies[i]->Render();
}
```

4. Next, in the Update function you should be updating the enemies using the method we defined earlier, although we have yet to implement this method. This should go right next to your UpdatePOWBlock function.

   ```
   UpdateEnemies(deltaTime, e);
   ```

5. Now let’s get that UpdateEnemies method implemented! Below is some example code for the UpdateEnemies method. This does not allow for Mario to hit and kill an enemy who is injured. This is something for you to add yourself. Again, read the comments and ensure you understand what is happening. It is easy to enter the code I have supplied, but you will need to create your own versions for other enemies or pickups.

   You will notice a few added functions too, Get and Set Alive and GetInjured. The GetInjured should be in the Koopa file and can simply return m_injured. As for the GetAlive and SetAlive, as these could be used on all characters eventually, you should make these in the Character files.

   a. Create a protected bool variable for m_alive and set to true in the contructor.
   b. A public function for SetAlive that takes a bool isAlive as a parameter and sets m_alive to equal this parameter.
   c. And a bool function, GetAlive (can be inline), that returns m_alive.

```
if (!m_enemies.empty())
{
	int enemyIndexToDelete = -1;
	for (unsigned int i = 0; i < m_enemies.size(); i++)
	{
		//check if the enemy is on the bottom row of tiles
		if (m_enemies[i]->GetPosition().y > 300.0f)
		{
			//is the enemy off screen to the left / right?
			if (m_enemies[i]->GetPosition().x < (float)(-m_enemies[i]-
>GetCollisionBox().width * 0.5f) || m_enemies[
				  i]->GetPosition().x > SCREEN_WIDTH - (float)(m_enemies[i]-
>GetCollisionBox().width * 0.55f))
				  m_enemies[i]->SetAlive(false);
		}
		//now do the update

		m_enemies[i]->Update(deltaTime, e);

		//check to see if enemy collides with player
		if ((m_enemies[i]->GetPosition().y > 300.0f || m_enemies[i]-
>GetPosition().y <= 64.0f) && (m_enemies[i]->
			GetPosition().x < 64.0f || m_enemies[i]->GetPosition().x >
SCREEN_WIDTH - 96.0f))
		{
			//ignore collisions if behind pipe
		}
		else
		{
			if (Collisions::Instance()->Circle(m_enemies[i], mario))
			{
				if (m_enemies[i]->GetInjured())
				{
					m_enemies[i]->SetAlive(false);
				}
				else
				{
					//kill mario
				}

			}
		}

		//if the enemy is no longer alive then schedule it for deletion
		if (!m_enemies[i]->GetAlive())
		{
			enemyIndexToDelete = i;
		}
	}

	//remove dead enemies -1 each update

	if (enemyIndexToDelete != -1)
	{
		m_enemies.erase(m_enemies.begin() + enemyIndexToDelete);
	}
}
```

1. Next, locate the DoScreenshake() function you wrote in a previous tutorial. Add some code to loop through the m_enemies vector and call the TakeDamage() function on each one. This will cause each Koopa in the scene to jump in the air and fall back down, upside down in their shell.
2. The last method we need to implement that we defined earlier if the CreateKoopa method. This method should create a new CharacterKoopa object (similar to how you created Mario and Luigi objects in SetUpLevel) and add it to the m_enemies vector using the following approach:

```
m_enemies.push_back(koopa);
```

3. Finally, inside your SetUpLevel method you can call the method we have just written to create new baddies. Call the CreateKoopa function twice to start the action. KOOPA_SPEED is a constant float variable stored in Constants.h set to 96.0f.

```
CreateKoopa(Vector2D(150, 32), FACING_RIGHT, KOOPA_SPEED);
CreateKoopa(Vector2D(325, 32), FACING_LEFT, KOOPA_SPEED);
```

Test and run your application to make sure everything has been done correctly. You should have some moving Koopas that flip when Mario hits the POW block and disappear off screen eventually.

Additional Work

1. Add code which checks if an enemy is injured when Mario collides with it. If so, set it to dead and remove it from the game.
2. Add a float variable to GameScreenLevel1, which will continually count down. When it reaches zero generate a new Koopa. Remember to reset the timer.
3. Implement code that turns your enemies around when they hit the boundaries of the screen, rather than walking off and being removed.
4. Repeat this tutorial and create a CharacterCoin class. There are some differences, which you should be aware of.
    a. There are 3 sprites on the coin sprite sheet.
    b. You need to animate the coin all the time for it to spin as it moves. The following code snippet will help, but you need to set up the variables in the header. Also, your ANIMATION_DELAY should be defined as a very small number, around 0.10f, but play with the value and see what works for you.

```
m_frame_delay -= deltaTime;
if(m_frame_delay <= 0.0f)
{
	//reset frame delay count
	m_frame_delay = ANIMATION_DELAY;

	//move the frame over
	m_current_frame++;

	//loop frame around if it goes beyond the number of frames
	if (m_current_frame > 2)
		m_current_frame = 0;
}
```

    c. Multiply the m_current_frame with m_single_sprite_w when rendering to get the left position of the coin sprite you need.

```
SDL_Rect portion_of_sprite = { m_single_sprite_w * m_current_frame,
0,m_single_sprite_w, m_single_sprite_h };
```

    d. The playable characters should kill the coin on collision. Add points for collecting coins.
5. Implement another enemy type from the sprites supplied on Blackboard.