UWE Bristol | University of the West of England

Games Technology (BSc) - Year 1
2020/2021
Games in C++

Angry Birds

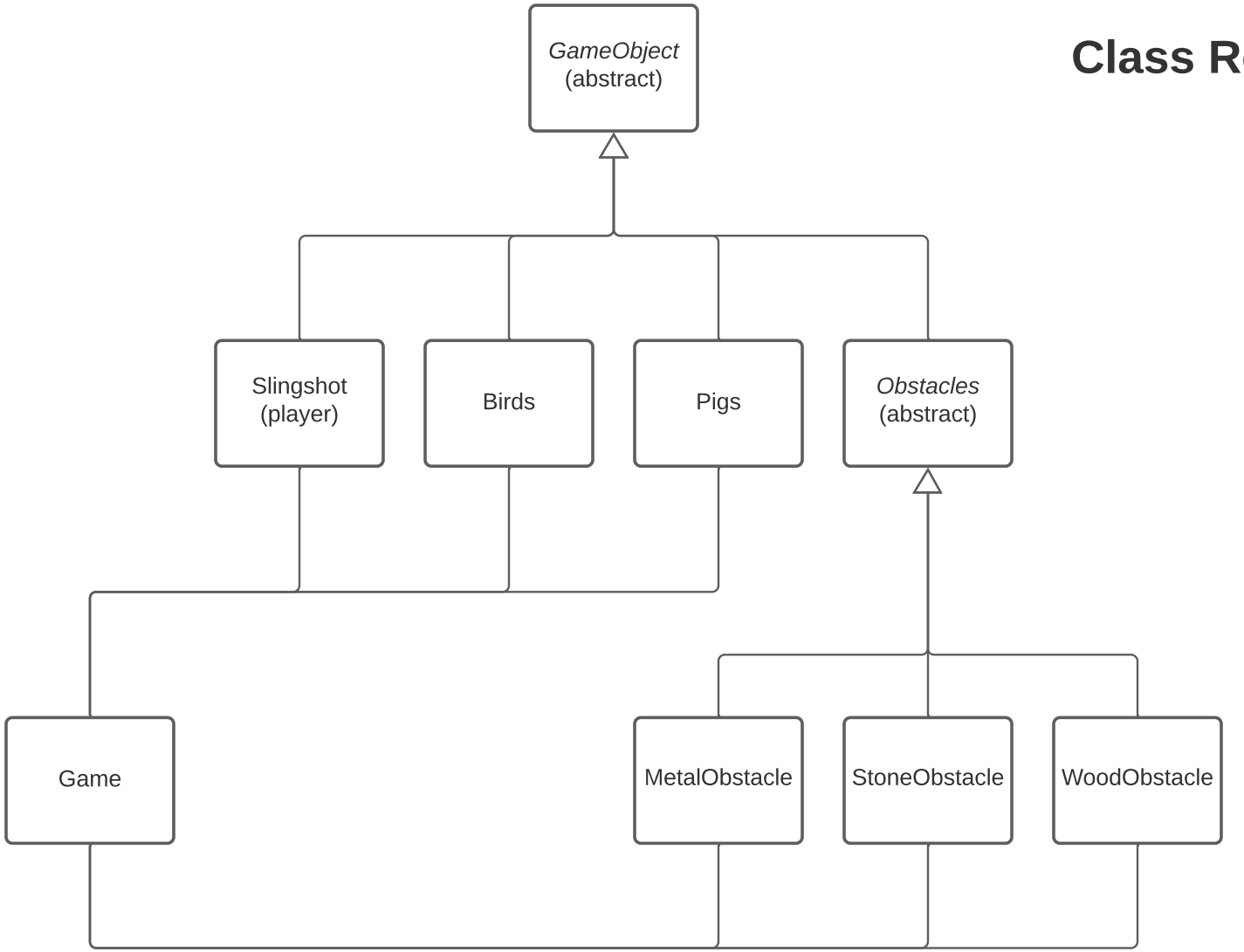
- Initial Class Diagram
- Class Diagram after implementation
- Brief discussion on changes in the Class Diagram
- Pseudo code: moving the bird object along a curve

Norbert Kupeczki - 19040948

Bristol - 28/04/2021

Angry Birds
Class Relationship Diagram
Initial plan

Norbert Kupeczki
19040948



Angry Birds
Class Diagram
Final implementation

Norbert Kupeczki
19040948

Buttons

#sprite: sf::Sprite*
#active_texture: sf::Texture
#inactive_texture: sf::Texture
#button_hover: sf::Sound*
#button_click: sf::Sound*
#active: bool

+Buttons()
+~Buttons()
+setTextures() <-Overloaded
+setSounds()
+getSprite(): sf::Sprite*
+mouseOver()
+activeBtn(): bool
+click()
+setPosition()

OptionsMenuButtons

#selected: bool

+mouseOver() <-Override
+switchTexture()
+isSelected(): bool

Options

-options_bg_spr: sf::Sprite
-options_bg_tex: sf::Texture
-background_music: sf::Music*
-MAX_VOLUME: const float
-music_volume: float
-music_on: bool
-options_active: bool
-accept_button: OptionsMenuButtons
-music_button: OptionsMenuButtons
-lvl1_button: OptionsMenuButtons
-lvl2_button: OptionsMenuButtons
-lvl3_button: OptionsMenuButtons
-volume_slider: VolumeSlider
-LevelSelected: enum
-level_selected: LevelSelected

+Options()
+~Options()
+initButtons()
+checkButtons()
+isSliderGrabbed(): bool
+moveSlider()
+initMusic()
+isOptionsActive(): bool
+enterOptionsMenu()
+levelSelected(): int
+mouseClicked()
+mouseReleased()
+renderOptions()

VolumeSlider

+MIN_POS: const int
+MAX_POS: const int
-grabbed: bool

+isGrabbed(): bool
+grabSlider()
+releaseSlider()
+moveSlider()
+mouseOver() <-Override

Game

-window: RenderWindow&
-font: sf::Font
-score_text: sf::Text
-score_endlevel: sf::Text
-background_spr: sf::Sprite
-background_tex: sf::Texture
-pause_spr: sf::Sprite
-pause_tex: sf::Texture
-help_menu_spr: sf::Sprite
-help_menu_tex: sf::Texture
-lvl1_tex: sf::Texture
-lvl2_tex: sf::Texture
-lvl3_tex: sf::Texture
-game_won_tex: sf::Texture
-gane_lost_tex: sf::Texture
-sling_full_tex: sf::Texture
-sling_left_tex: sf::Texture
-bird_tex: sf::Texture
-pig_tex: sf::Texture
-menu_hover_bfr: sf::SoundBuffer
-menu_click_bfr: sf::SoundBuffer
-slingshot_launch: sf::SoundBuffer
-pig_squeek: sf::SoundBuffer
-GameState: enum
-game_state: GameState
-game_paused: bool
-level_set: bool
-help_open: bool
-NUMBER_OF_PIGS: const int
-GRAVITY: const float
-MAX_DIST: const float
-pigs_left: int
-elapsed_time: float
-slingshot: Slingshot
-options: Options(window)
-level_manager: LevelManager
-pigs: Pig*
-bird: Birds
-lives: Birds*
-play_button: Buttons
-help_button: Buttons
-options_button: Buttons
-menu_button: Buttons
-quit_button: Buttons

Game()
~Game()
init(): bool
update()
render()
mouseClicked()
mouseReleased()
keyPressed()
loadTextures() <-Overloaded
loadSound()
initTextures()
initButtons()
initSounds()
initFont()
updateText()
setButtonPosition()
initLevel()
updateMenus()
updateInGame()
checkButtons()
getDistance(): float
moveBird()
launchBird()
collisionCheck()
checkEndOfGameConditions()
drawPigs()
drawButtons()
drawLives()
resetBird()
resetGame()

Pig

-value: int
-pig_squeek: sf::Sound*

+Pig()
+~Pig()
+setSound()
+squeek()
+getValue(): int

GameObject

#visible: bool
#sprite: sf::Sprite*

+GameObject()
+~GameObject()
+setTexture()
+getSprite()
+setVisibility()
+isVisible(): bool

Birds

#grabbed: bool
#launched: bool
#bird_vector: Vector2
#bird_start:Vector2
#speed: float

+Birds()
+~Birds()
+isGrabbed(): bool
+setGrabbed()
+isLaunched():bool
+setLaunched()
+setSpeed()
+getSpeed(): float
+setVector()
+getVector(): Vector2
+normaliseVector()
+setStart()
+getStart(): Vector2

LevelManager

-metal_obstacle: MetalObstacle*
-sonte_obstacle: StoneObstacle*
-wood_obstacle: WoodObstacle*
-boulder: Boulder
-number_of_metals: int
-number_of_stones: int
-number_of_woods: int
-metal_tex: sf::Texture
-stone_tex: sf::Texture
-wood_tex: sf::Texture
-boulder_tex: sf::Texture

+LevelManager()
+~LevelManager()
+getPigCoords(): sf::Vector2f
+levelTexture(): sf::Texture&
+createObstacles()
+deleteObstacles()
+collisionCheck(): bool
+checkObstacles(): bool
+renderObstacles()
+getBoulder(): sf::Sprite*
+moveBoulder()
+resetBoulder()
-getLvl1PigCoords: sf::Vector2f
-getLvl2PigCoords: sf::Vector2f
-getLvl3PigCoords: sf::Vector2f
-loadTexture()

Slingshot

+MAX_LIVES: const int
-score: int
-lives: int
-NUM_OF_CIRCLES: const int
-circles[ ]: sf::CircleShape
-sprite_left: sf::Sprite*
-launch_pos: sf::Vector2f
-slingshot_launch: sf::Sound*

+Slingshot()
+~Slingshot()
+setPostition()
+getSpriteLeft(): sf::Sprite*
+setTextures()
+setSound()
+launch()
+getLaunchPos(): sf::Vector2f
+drawTrajectory()
+trajectory()
+setLives()
+loseLife()
+getLives(): int
+addScore()
+resetScore()
+getScore(): int
-initTrajectory()
-getDistance(): float

Boulder

-INIT_SPEED: const float
-speed: float

+Boulder()
+moveBoulder()
+resetSpeed()

WoodObstacle

StoneObstacle

MetalObstacle

+MetalObstacle()

Obstacles

#destructible: bool

+Obstacles()
+destroyObstacle()
+isDestructible(): bool

A brief discussion of changes in class relations

According to the original plan, an abstract **GameObject** class has been planned to provide as a blueprint to other derived classes, the slingshot, the pigs, and the birds. There was another abstract class planned to be derived from the **GameObject** class, called Obstacles, what supposed to have 3 children classes, one for each obstacle type: metal, stone, and wood obstacles. The original idea was that the **Game** class will manage all the game objects, but as more and more features were added, this class started to become hard to read, and complicated to extend.

Compared to the initial plan, the extended class diagram shows minor changes to how the classes relate to each other. As the code expanded and became more complex, to improve readability, new classes have been added to encapsulate certain functions and organise them into logical units.

One such addition is the **Options** class, its role to store and manage data about the game and sound settings, also to manage the rendering of its related objects, the buttons and the menu background. Upon starting a new game, before the level is initialised, the **levelSeleced()** class method is called, returning the index of the selected level, which is then passed to the **LevelManager** class to set up the level.

The **LevelManager** class has been added to encapsulate all attributes and methods to manage and render the obstacles on a level to the active window. The level index returned from the **Options** class is forwarded to the **LevelManager**, where the **createObstacles()** method based on this index creates the obstacle object arrays and then positions the objects. The other important task of this class is to remove all objects when the game is over and free up the dynamic memory using the **deleteObstacles()** class method.

The **Buttons** class is a minor addition to provide a framework for all button objects in the game. This class is used by the **Game** class to manage its buttons. For the **Options** class, a child class has been added called **OptionsMenuButtons** that extends the functionality of the parent class with additional methods. There is another derived class from the **OptionsMenuButtons** class called **VolumeSlider**, the only use of this class at the options menu to manage the volume of the background audio.

The last addition to the original plan was the **Boulder** class, which is derived from the **Obstacles** class. When the addition of optional features was considered, an object that can interact with other game elements, but not controlled by the player became an option. The purpose of this class is to manage the movement of a boulder that can crush the pig objects.

Looking back at the original plan after finishing with the coding, I can say it was a good starting point, but with time as all the basic requirements from the brief has been implemented, the additional features made a few changes in the class relations and compositions necessary to maintain the readability and make it easier to extend the codebase. By adding all the above to the initial plan I gained a better understanding of how encapsulation can help to create a better, clearer code.

Pseudo code for moving the bird, with explanation

void launchBird() ← *This function is called upon releasing the left mouse button, if* ***(bird.grabbed == true)***

{

Subtract vector pointing to B from vector pointing to A, to get **vector_d**.

Calculate **strength** based on the magnitude of **vector_d**.

Normalize **vector_d**.

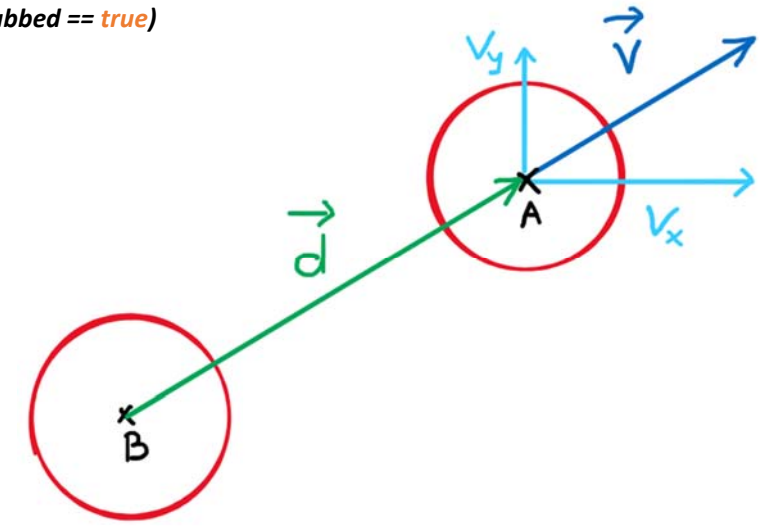
Multiply **vector_d** by **strength** and an arbitrary **SPEED** constant to get **vector_v**.

Set **bird.start_position** vector

Set **bird.launched** bool to **true**

Set **elapsed_time** to **0.0**

}



If (bird.launched) ← *This goes to the update() function*

{

Calculate new **x_position** = **vector_v.x** * **elapsed_time**

Calculate new **y_position** = **vector_v.y** * **elapsed_time** - **GRAVITY** * **elapsed_time**^2

Set bird's sprite position = **{start_position.x** + **x_position, start_position.y** + **y_position}**

Increment **elapsed_time** by **deltaTime**

}

Note:

In the above pseudo code, **A** is the starting position vector of the bird when it sits in the slingshot, **B** is the position vector from where the player launches it.

SPEED and **GRAVITY** are arbitrary constants.

bird.grabbed is true, if upon pressing the left mouse button, the cursor intersects the bird sprite.