README, PROJECT - OOP 2019/20

**“DIGGER”**

by Akiva Stern – 320818008 and Rachel Wolfe – 337816656

***\*\*\*find at the end of this document everything that has been changed since the presentation\*\*\****

**Overview:**

This project is our expression of the game “Digger”.

The game is run by defining an object of type Controller. Then this object calls the function to play the game.

The objective of the game is to complete all the levels without losing all of his/her lives. The player does this by collecting all the diamonds in each level, while trying to avoid obstacles, including the monsters, who will “eat” the player if he is caught. The level has other variables such as a possible time limit, points to try and get, stones allowed to eat, and bonuses throughout. The player moves throughout each level by using the direction keys on the keyboard and the spacebar to stop the movement in the given direction.

**Design:**

The main will create an object Controller, which will manage the game. The main will then call to setup the level with this object. The following are the rest of the objects in the program:

**Board:** the board holds the textures of all the **GameObjects** within the game and it determines if the digger ever runs into a collision of any sort. It holds the matrix, which is the game board, in its entirety, with all the objects in their allocated locations.

These objects are held as **ImmobileObject\*** allowing us to use some nice *polymorphism* to activate their appropriate functions, varying on the actual object in real-time.

It initiates the game by running through the received matrix and translating each character to what it represents. As it converts the monsters, it prints one out of every three monsters as a **StupidMonster** and the other two as **SmartMonsters** within a vector of pointers to either type of **Monster**. It confirms there are no current collisions and draws the board in the window for the user to see and begin playing the game. At the end it then erases everything that it built/drew before finishing and moving on to the next screen or ending the game.

**Controller:** the controller, just like its name, controls the whole game. Though each object type that it holds handles its own functionality, the controller manages the game a whole.

The controller holds the amount of each **gameObject** (including the digger and the vector of monsters) and where each object is at the beginning of each level, as per the stats of the game that go according to that. It also holds the number level, time counter, points and allowed stones as the game goes on. It determines what to do when the player loses or wins each level or ends the game. In addition to all the above, it is where the textures for the backgrounds and opening/closing screens of the game are stored for the window to use.

While the game is being played, it determines when the player interacts with the game window, to move the digger or to close the window altogether if the player chooses to exit. It checks if the level needs to be reset or restart and if the level was won. It is responsible also for the display of the points, lives, rocks allowed and time remaining.

**GameObject:** this is the base class for all objects in the game both moving and stationary. It holds their draw, location and sprite functions, as well as its own checker for collisions which is inherited by each of the objects below that is uses it (monster, diamond, rock and wall).

**ImmobileObject:** Inherits from **GameObject** and adds the function to check if there was a collision with a MovingObjectof any sort which is used with the wall. It draws each of the objects and returns when called if the **MovableCharacter** collided with a wall.

**MovableCharacter:** Inherits from **GameObject** and adds on functions for use in the **Monster** and **Digger** classes. The class has functions to set the directions for moving and an adjacent get function for when needed. It has its own version of the move which is only used by classes than inherit from it. This class also hold a function to determine if the character has gone out of bounds or gotten stuck on a wall.

**Bonus:** Inherits from **ImmobileObject** and produces different types of bonuses for each round and for their images to be printed out in the location during the building of the level and the points and corresponding bonuses to be added as expected.

**BonusPoints:** Inherits from **Bonus** and implements the collision consequence function by adding points for the player.

**BonusStones:** Inherits from **Bonus** and implements the collision consequence function by allowing the digger to eat additional stones.

**BonusTime:** Inherits from **Bonus** and implements the collision consequence function by adding time for the player on the level.

**Diamond:** Inherits from **ImmobileObject** and implements the collision consequence function by removing the diamond from the game board and adding points for the player and checking if the level was won yet.

**Star:** Inherits from **ImmobileObject** and implements the collision consequence function by removing the star from the board and adding points.

**Stone:** Inherits from **ImmobileObject** and implements the virtual collision consequence function by deleting the stone from the board and decreasing the amount of allowed stone the digger can eat.

**Wall:** Inherits from **ImmobileObject** and implements the virtual collision consequence function by reacting when a **MovableCharacter** runs into it to prevent them from continuing movement.

**Digger:** Inherits from **MovableCharacter**, draw and move. The digger also receives the movements that the player enters (with the arrows and space on the keyboard). The inherited move function checks if the player runs into a wall of off the board and stop the movement upon arrival and waits for the player to respond until starting movement again.

**Monster:** Inherits from **MovableCharacter**, draw and move. It creates its own virtual chase to be inherited from both types of monster and hold a set and get for member eaten. To update and check if the monster and digger run into one another during game play. The inherited move function checks if the monster runs into a wall and changes direction or stop accordingly. The Controller will hold a vector of Monster\*, allowing us to hold either of the two below monsters in the same data structure and we use *polymorphism* to the appropriate functions. The monsters move half as fast as the digger moves.

**SmartMonster:** Inherits from **Monster** and adds its own implementation of the inherited function chase. The monster checks where it is in relation to the digger and moves up or down and then left or right each iteration to get closer to the digger in order to eat it so the player loses the game before getting all the diamonds.

**StupidMonster:** Inherits from **Monster** and adds its own implementation of the inherited function chase. The direction is chosen by randomly generating a number below eight and going in the direction assigned to that number in the switch cases: (up, down, right, left, and each of the diagonals). It then checks if it caught the digger and repeats.

**BonusInfoDisplayer:** A data structure dynamically holding and then drawing the number that any bonus that the digger may have eaten is worth(i.e. a bonus time of **25 seconds**). This info per bonus is displayed briefly.

**StatsDisplay:** Draws to the side of the board the stats of the games with the level, amount of points, amount of allowed rocks to eat and time left in level. They are displayed and updated as needed as the game keeps running and the player advances and continues playing the game.

**Board.txt File Format:**

The board is surrounded by walls and starts with a list of integers in this order: amount of rows, and then columns on the given board, how many stones the digger is allowed to eat before losing a life, time allowed for level (-1 = no time limit for specific level).

The following are symbols shown on the board.txt file and what they are converted into:  
  
/ - Digger  
! - Monster  
D - Diamond  
@ - Stone  
# - Wall

In order to change the game playing boards the current ones may be replaced by placing a new file by the name BoardX.txt in the resource folder (where X is the number level). To add on more levels to the game, just add another boardX.txt file to the resources folder in the project and this file **MUST BE added to the CMakeLists file of the resources folder.**

**Files in Program:**

**\*\*\*All these class objects were described above\*\*\***

Headers for all the classes:

Board.h – header for class Board

Bonus.h – header for class Bonus

BonusInfoDisplayer.h – header for class BonusInfoDisplayer

BonusPoints.h – header for class BonusPoints

BonusStones.h – header for class BonusStones

BonusTime.h – header for class BonusTime

Controller.h – header for class Controller

Diamond.h – header for class Diamond

Digger.h – header for class Digger

GameObject.h – header for class GameObject

ImmobileObject.h – header for class ImmobileObject

Monster.h – header for class Monster

MovableCharacter.h – header for class MovableCharacter

SmartMonster.h – header for class SmartMonster

StatsDisplay.h – header for class StatsDisplay

Stone.h – header for class Stone

StupidMonster.h – header for class StupidMonster

Wall.h – header for class Wall

Non-object header file:

Constsants.h – holds all the consts in the program

For each header file there is an adjacent CPP file for the function implementations:  
(except Constsants.h)

Board.cpp, Controller.cpp, StatsDisplay.cpp, GameObject.cpp, ImmobileObject.cpp, Diamond.cpp, Stone.cpp, Wall.cpp, Bonus.cpp, BonusPoints.cpp, BonusStones.cpp, BonusTime.cpp, BonusInfoDisplayer.cpp, MovableCharacter.cpp, Digger.cpp, Monster.cpp, StupidMonster.cpp, SmartMonster.cpp

And of course, also the main.cpp holds the initiation of the controller of the game

Resources:

<object>.png: each object that is printed on the screen has a png file.

<background>.jpg: the backgrounds of the program are in jpg format

<sound>.wav: all sound effects and music is from a wav file

There are also BoardX.txt files that hold the game boards to play in the resources

**Data Structures:**

As described in the design above, are all the main data type structures used.

Notably, the controller which carries all the info, and the matrix of movable objects that is held in the board.

**Potential Bugs:**

It is important to note that if the dimensions of the board are too large for your computer screen. The settings are initially 1600x800 pixels but may be changed to be smaller in order to fit in the screen in Constsants.h with the consts: SCREEN\_HEIGHT and SCREEN\_WIDTH.

**Algorithms:**

The stupidMonsters randomly generate a number from zero to three and goes in the assigned direction until a new direction is set for it in the sprite.

The smartMonsters search for the digger and follow him right left, up or down (or diagonally) until they get to the digger. If the player is tactful, he will maneuver around the rocks and walls so that the monsters get stuck behind them and then he can succeed at eating all the diamonds before getting eaten himself by one of the monsters.

**MORE COMMENTS:**

The Monsters’ speed can be set in the Constsants.h file, and similarly the diggers. In order to create a comfortable yet challenging enough gameplay, we decided on a large enough difference between the speed of the monsters and the digger, while digger is faster. Because the smart monsters can move easier in non-axis directions, toward the digger, than the digger himself can move (as it is humanly difficult to move diagonal with just horizontal and vertical direction keys), the smart monsters have an advantage over him. This is balanced out by the fact that the Digger is faster.

Another note regarding our design of the Monsters;

A Smart Monster (blue colored) may be “manipulated” into “chasing a wall” if the player (Digger) can strategically and tactfully cause the monster to chase him with a wall or stone in between. Until the Digger is facing a direction which is not blocked by this static object, the Smart Monster will appear locked in its place, when really this is a helpful strategy to avoid the monster and hopefully win the game!

**What has changed since the presentation:**

The graphics in the game were changed slightly, the opening and winning/losing pages, and the digger is now Yoshi, he is animated and changes directions as the player presses different keys to move. The monsters also change direction by themselves as they walk right or left.

As per our session with Michal and Yechezkel, it was suggested to us the following which we have implemented:

-Make the “m\_lives” member, belonging to the Digger, as this makes for more generic code, as opposed to where we put it before - in the Controller. Done.

-Originally we had a too revealing (חושפני) function in the Board, which returned the m\_matrix itself. We have changed this, as per their comments, to just return a specific object in the board, upon request of its index (i, j). It is also const, thereby restricting any changes to the object!

**WE HOPE YOU ENJOYED THE GAME :)**