Requirements Specification for Block Game

By Adam Kennedy

Contents

[Analysis 3](#_Toc129901000)

[Description of the problem 3](#_Toc129901001)

[Scope 3](#_Toc129901002)

[Boundaries 4](#_Toc129901003)

[Constraints 4](#_Toc129901004)

[UML use case diagram 5](#_Toc129901005)

[Required specification 6](#_Toc129901006)

[End-user requirements 6](#_Toc129901007)

[Functional requirements 6](#_Toc129901008)

[Project plan for each stage 7](#_Toc129901009)

[Gantt chart 11](#_Toc129901010)

[Design 12](#_Toc129901011)

[Design of Project 12](#_Toc129901012)

[Structure Diagram 12](#_Toc129901013)

[Pseudocode that fulfils functional requirements 12](#_Toc129901014)

[Main game loop 12](#_Toc129901015)

[Design refinements 13](#_Toc129901016)

[Pseudocode for the rest of the functional requirements: 14](#_Toc129901017)

[Pseudocode for End User Requirements 19](#_Toc129901018)

[Design of Advanced Higher Concepts 21](#_Toc129901019)

[UML Class Diagrams 22](#_Toc129901020)

[Design of Array of Objects 23](#_Toc129901021)

[Design of integration 25](#_Toc129901022)

[Entity Relationship Diagram 25](#_Toc129901023)

[Data Dictionary 25](#_Toc129901024)

[Query Designs 25](#_Toc129901025)

[User-Interface 26](#_Toc129901026)

[Playing UI 26](#_Toc129901027)

[Game Over UI 26](#_Toc129901028)

[Implementation 27](#_Toc129901029)

[AH Concepts Implementation 27](#_Toc129901030)

[Bubble Sort 27](#_Toc129901031)

[Array of Objects 27](#_Toc129901032)

[Implemented Integration 28](#_Toc129901033)

[Implemented UI 30](#_Toc129901034)

[Implement Functional requirements 31](#_Toc129901035)

[Implemented End-User Requirements 34](#_Toc129901036)

[New Skills/Knowledge 35](#_Toc129901037)

[References 35](#_Toc129901038)

[Log of ongoing Testing 36](#_Toc129901039)

[Testing 36](#_Toc129901040)

[Plan for Final Testing 36](#_Toc129901041)

[Evidence of Requirements Testing 37](#_Toc129901042)

[Screenshots Showing Inputs and Any Errors Generated for My Program 37](#_Toc129901043)

[Screenshots Showing Successful Implementation of Advanced Higher Algorithms 38](#_Toc129901044)

[Screenshots Showing Successful Implementation of SQL Queries 39](#_Toc129901045)

[Stub, Diver and Debug Print Statements Testing 39](#_Toc129901046)

[Testing database server connect function: 39](#_Toc129901047)

[Print Debug Statements 40](#_Toc129901048)

[Full System Testing 40](#_Toc129901049)

[By Programmer 40](#_Toc129901050)

[By Beta Testers 41](#_Toc129901051)

[Testing with persona and test cases 42](#_Toc129901052)

[Results of Test Cases 42](#_Toc129901053)

[Evaluation 43](#_Toc129901054)

[Evaluation of solution 43](#_Toc129901055)

[Fitness for Purpose 43](#_Toc129901056)

[Evaluation of Solution in terms of: 43](#_Toc129901057)

[Efficiency 43](#_Toc129901058)

[Future Maintainability 43](#_Toc129901059)

[Robustness 43](#_Toc129901060)

[Appendix 44](#_Toc129901061)

[Design of Modules for Block Game: 44](#_Toc129901062)

[App Module: 44](#_Toc129901063)

[MysqlServer Module: 46](#_Toc129901064)

[MySQLServerConfig Module: 48](#_Toc129901065)

[Game Module: 49](#_Toc129901066)

[endgamecontroller Module: 53](#_Toc129901067)

[Blocks Module 56](#_Toc129901068)

[Config Module: 59](#_Toc129901069)

# Analysis

## Description of the problem

My project will be a game called Block Game that will consist of mostly classes utilizing OOP concepts with an array of objects, and it will also make use of the Bubble sort algorithm. The game will store the high score in a database using EasyPHP, integrating the program and database. During the game the player will have to use W, E, A, S, D keys to rotate and move the shapes that drop from the top of the screen. The blocks will be unable to move past the left and right side of the board and will be rotatable and movable if the block doesn’t collide with anything. The player will gain points for creating entire rows of blocks that will then be cleared, giving the player points and a multiplier will be applied the more rows cleared at once. The game will continue forever until the player loses(when the blocks stack up to the top of the screen). When the player loses, they will be presented with the top 5 highs sores and their score with a place to enter three letters that will then be used to store their score in the database.

The end-users of my game will be for teenagers aged 16 to 19 that like puzzle game, but it will also be made to be usable by anyone who likes puzzles.

My project meets all the requirements set out in the advanced higher computer science course because it will be suitable for my end-user and will be capable of validating all user inputs. My project will interface with a database that stores the users chosen 3 letter name and score. This data will be inserted and queried using MySQL and hosted in a database using EasyPHP. This data will be used to show the top 5 high scores when games end. This leader board of the top 5 players will be sorted from highest to lowest using a sorting algorithm.

### Scope

The scope of my project will include:

* My project will be completed by Friday 17th of March 2023
* By the end of my project, I will have a completed and detailed document containing the analysis, design, implementation, testing and evaluation of my project.
* By the end of my project, I will have a working game that meets the functional and end-user requirements.

### Boundaries

My completed project will contain:

* A detailed document containing all stages in the development of my project:
  + Analysis – this section will contain all the concepts and limits of my project.
  + Design – this section will contain all the important highlights in the design of my project such as advanced higher concepts.
  + Implementation – this section will contain all of my implemented AH concepts, requirements and integration. It will show all the code required to implement these
  + Testing – This section will show how all the components of my program have been tested and work together
  + Evaluation – This section will cover whether my program is fit for purpose, robust and maintainable.
* A fully completed project that meets all the functional and end-user requirements. This project will make use of arrays of objects as well as a sorting algorithm from the advanced higher course.

### Constraints

The constraints on my project are:

* I must not be in breach of GDPR or the Copyright, Designs and Patents Act 1988
* I will only have 2 months to complete the project.
* I complete this project without spending any money as I am using software that is fully licensed by the school as well as free software.
* I will have to compromise on graphics and polish due to time constraints.
* I will be using Python version 3.6.2 and Pygame as well as MySQL Connector for making my game and interfacing with the database due to my experience with these tools.

## UML use case diagram

Diagram

Description automatically generated

## Required specification

### End-user requirements

1. Player must be able to move blocks left, right and down(drop) if valid.
2. Player must be able to rotate blocks either left or right if new rotation can fit within the space.
3. Player must be able to enter 3 letters to represent them and their high score
4. Player must be able to see the top 5 previous players at end of game

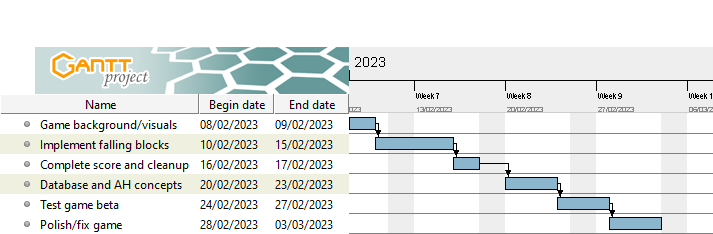
### Functional requirements

1. Game must run at 60 FPS
2. Game must start immediately when ran.
3. Once game has started blocks have to spawn one at a time at the max height of the board.
4. Blocks must be bound to the playable area.
5. Interface must show score , falling block and play area.
6. Database has got to be able to store multiple people of the same name with the top 5 highest scores.
7. Once the game ends the top 5 high scores must be shown with 3 letters entered by the player that got that score.
8. The game should update the players score when they clear a line, giving a bonus for how many lines that are cleared at once.
9. Player movements should be validated.
10. Player name entry after game is over should be validated.

## Project plan for each stage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Sub Task | Time(days) | resources | people |
| Analysis | Determine functional and end-user requirements | 2 | Microsoft word | programmer |
| Design | top level design with data flow | 3 | Microsoft word, diagrams.net | programmer |
| Refinements of advanced higher concepts, functional requirements, and input validation | 3 | Microsoft word | programmer |
| Use UML class diagrams to show design of all classes in program | 2 | Microsoft word, diagrams.net | programmer |
| Design structure and data types of arrays of records | 3 | Microsoft word, diagrams.net | programmer |
| Create wireframes to show user interface | 2 | Microsoft word, diagrams.net | programmer |
| Design | Entity relationship diagram | 2 | Diagrams.net | programmer |
| Data dictionary – use SQL attributes and indicate input validation | 2 | Microsoft word | programmer |
| Implementation | Show important program code | 4 | Computer, python | programmer |
| Screenshots of UI | 0.5 | Computer, python | programmer |
| SQL code | 2 | computer, python | programmer |
| Screenshots to show that structure of implemented table matches the design | 4 | computer, python, easyphp | programmer |
| Research and development of new skills | 5 | Computer, python, easyphp | programmer |
| Testing | Log of ongoing testing | 2 | Computer and beta testers | Classmates, programmer |
| Use feedback from beta testers to fix/polish game | 4 | computer, python, easyphp | Classmates, programmer |
| Create final plan: including tests for all end-user and functional requirements.  A description of tests that will be carried out  A description of personas with a list of test cases that will be used to test the solution. | 4 | Microsoft Word | programmer |
| Evidence of tests for all functional requirements. Evidence could be screenshots | 2 | Windows screenshot, python | programmer |
| Test solution with persona and test cases. Describe the results of each test case using table. | 3 | Python, Microsoft word | Classmates, programmer |
| Evaluation | Produce report for fitness for purpose:  How close does the solution match all the requirements in requirements in requirements specification  The results of testing | 3 | Microsoft word | programmer |
| Document the future maintainability and robustness of the solution | 3 | Microsoft word | programmer |

## Gantt chart



# Design

## Design of Project

## Structure Diagram

Diagram

Description automatically generated

## Pseudocode that fulfils functional requirements

### Main game loop

FR 1:

SET application TO new instance of app()

CALL application.Main()

Initialises and calls the application class and main game loop.

Application.Main() calls:

Refinement

1. PROCEDURE Main(self)

2. DECLARE run AS INITIALLY TRUE

3. WHILE run IS TRUE DO

4. IF self.game.restart IS TRUE THEN

5. self.game = NEW Game(self)

6. END IF

7. CALL CheckForEvents()

8. CALL Update()

9. CALL Draw()

10. END WHILE

11. END PROCEDURE

Explanation:

Refinement for main game loop.

### Design refinements

Refinement(7.x)

7.1 PROCEDURE CheckForEvents(self):

7.2 SET self.pressedKey = None

7.3 SET self.animTrigger = False

7.4 SET self.fastAnimTrigger = False

7.5 FOR EACH event IN pg.event.get() DO

7.6 IF event.type = pg.QUIT THEN

7.7 CALL pg.quit()

7.8 CALL sys.exit()

7.9 END IF

7.10 IF event.type = pg.KEYDOWN THEN

7.11 CALL self.game.Controls(pressedKey=event.key)

7.12 SET self.pressedKey = event.key

7.13 END IF

7.14 IF event.type = self.userAnimEvent THEN

7.15 SET self.animTrigger = True

7.16 END IF

7.17 IF event.type = self.userFastAnimEvent THEN

7.18 SET self.fastAnimTrigger = True

7.19 END IF

7.20 END FOR

7.21 END PROCEDURE

Explanation:

This procedure controls the updates for the game

FR 2:

Refinement(8.x)

8.1 PROCEDURE Update(self)

8.2 CALL self.game.Update()

8.3 CALL self.clock.tick(FPS)

8.4 END PROCEDURE

Explanation:

This procedure updates the game state and keeps the game running at 60 frames per second

Refinement(9.x)

9.1 PROCEDURE Draw(self)

9.2 CALL self.screen.fill(FIELD\_COLOR)

9.3 CALL self.game.Draw()

9.4 IF NOT(self.game.gameOver) THEN

9.5 CALL self.display.blit(self.screen, FIELD\_OFFSET + (0, 0))

9.6 END IF

9.7 CALL pg.display.flip()

9.8 END PROCEDURE

Explanation:

This function draws everything to the screen.

Within the surface calls of the main game loop functional requirements 1 and 2 fulfilled.

### Pseudocode for the rest of the functional requirements:

FR 3(spawning blocks):

1. FUNCTION GetArrayOfBlocks(self):

2. SET positions TO SHAPES[self.shape]

3. SET blocks\_array TO empty list

4. FOR EACH pos IN positions DO

5. ADD new instance of Block(self, pos) TO blocks\_array

6. END FOR

7. RETURN blocks\_array

8. END FUNCTION

FR 4(blocks must be bound to playable area):

1. FUNCTION IsCollide(self, pos):

2. SET x TO integer value of pos.x

3. SET y TO integer value of pos.y

4. IF (0 <= x < FIELD\_W) and (y < FIELD\_H) and ((y < 0) or (not self.shape.game.fieldArray[y][x])):

5. RETURN False

6. ELSE:

7. RETURN True

8. END IF

9. END FUNCTION

FR 5(interface must show score , falling block and play area):

1. PROCEDURE Draw(self)

2. IF NOT(self.gameOver) THEN

3. CALL self.DrawOverlay()

4. CALL self.DrawGrid()

5. CALL self.blockGroup.draw(self.app.screen)

6. CALL self.DrawScore()

7. ELSE

8. CALL self.endGameController.Draw()

9. END IF

10. END PROCEDURE

FR 6(store top 5 players scores, names can be the same):

1. SET TABLES['highscores'] =(

2. 'CREATE TABLE HighScores('

3. ' id varchar(3) NOT NULL,'

4. ' score INT NOT NULL)'

5. )

FR 7(implementation of game ending, shows player score and top 5 players):

1. PROCEDURE Draw(self):

2. CALL self.game.app.display.fill((0, 0, 0))

3. CALL self.game.app.screen.fill((0, 0, 0))

4. CALL self.DrawHighscores()

5. CALL self.DrawPlayerScore()

6. END PROCEDURE

Refinements

4.1 PROCEDURE DrawHighscores(self)

4.2 SET textHeight TO 0

4.3 SET scoreHeaderText TO self.DrawText(HIGHSCORE\_HEADER\_OFFSET, HIGHSCORE\_HEADER,

textColor=TEXT\_COLOR, antialias=True)

4.4 FOR EACH previousPlayerInfo IN self.prevPlayerTop5

4.5 FOR EACH text IN self.prevPlayerTop5[0:i]

4.6 ADD text.get\_height() TO textHeight

4.7 END FOR

4.8 SET blitPos TO (HIGHSCORE\_HEADER\_OFFSET + (0, scoreHeaderText.get\_height()) + (0,

textHeight))

4.9 self.game.app.display.blit(previousPlayerInfo, blitPos)

4.10 SET textHeight TO 0

4.11 END FOR

4.12 END PROCEDURE

5.1 PROCEDURE DrawPlayerScore(self)

5.2 SET textHeight TO 0

5.3 SET playerHeader TO self.DrawText(PLAYER\_SCORE\_HEADER\_OFFSET, PLAYER\_HEADER,

textColor=TEXT\_COLOR, antialias=True)

5.4 SET player TO self.GetText(self.player, TEXT\_COLOR, antialias=True)

5.5 SET score TO self.GetText(str(self.game.score), TEXT\_COLOR, antialias=True)

5.6 SET playerInfoTup TO (player, score)

5.7 FOR EACH playerInfo IN playerInfoTup

5.8 FOR EACH text IN playerInfoTup[0:i]

5.9 ADD text.get\_height() TO textHeight

5.10 END FOR

5.11 SET blitPos TO (PLAYER\_SCORE\_HEADER\_OFFSET + (0, playerHeader.get\_height()) + (0,

textHeight))

5.12 self.game.app.display.blit(playerInfo, blitPos)

5.13 SET textHeight TO 0

5.14 END FOR

5.15 END PROCEDURE

FR 8(player score updated correctly):

1. FUNCTION ClearLines(self)

2. SET lines TO 0

3. SET row TO FIELD\_H - 1

4. FOR y IN RANGE(FIELD\_H - 1, -1, -1)

5. FOR x IN RANGE(FIELD\_W)

6. SET self.fieldArray[row][x] TO self.fieldArray[y][x]

7. IF self.fieldArray[y][x] THEN

8. SET self.fieldArray[row][x].pos TO pg.math.Vector2(x, row)

9. END IF

10. END FOR

11. IF SUM(MAP(BOOL, self.fieldArray[y])) < FIELD\_W THEN

12. SET row TO row - 1

13. ELSE

14. FOR x IN RANGE(FIELD\_W)

15. SET lines TO lines + 1

16. SET self.fieldArray[row][x].living TO False

17. SET self.fieldArray[row][x] TO 0

18. END FOR

19. END IF

20. END FOR

21. RETURN lines

22. END FUNCTION

lines passed to:

PROCEDURE UpdateScore(self, clearedLines)

SET self.score TO self.score + (SCORE\_BASE \* (clearedLines \*\* SCORE\_MULT))

END PROCEDURE

FR 9:

1. PROCEDURE Controls(self, pressedKey)

2. SET self.lastKeyPressed TO pressedKey

3. IF pressedKey == pg.K\_a THEN

4. CALL self.shape.Move('left')

5. ELSE IF pressedKey == pg.K\_d THEN

6. CALL self.shape.Move('right')

7. ELSE IF pressedKey == pg.K\_s THEN

8. SET self.speedUp TO True

9. ELSE IF pressedKey == pg.K\_w THEN

10. CALL self.shape.Rotate('left')

11. ELSE IF pressedKey == pg.K\_e THEN

12. CALL self.shape.Rotate('right')

13. END IF

14. END PROCEDURE

FR 10:

1. PROCEDURE GetPlayerIdInput(self)

2. SET pressedKey TO self.game.app.pressedKey

3. IF (pressedKey != pg.K\_BACKSPACE) AND (LEN(self.player) < 3) AND (pressedKey != NONE) AND

(pressedKey >= 97 AND pressedKey <= 122) THEN

4. SET self.player TO self.player + CHR(pressedKey).UPPER()

5. ELSE IF pressedKey == pg.K\_BACKSPACE THEN

6. SET self.player TO self.player[:LEN(self.player) - 1]

7. ELSE IF (LEN(self.player) == 3) AND (pressedKey == pg.K\_RETURN) THEN

8. IF NOT(self.prevPlayerTop5 == 5) THEN

9. CALL self.game.app.mySQLServer.InsertStatement('INSERT INTO highscores(id, score)

VALUES("' + STR(self.player) + '", "' + STR(INT(self.game.score)) + '")')

10. SET self.game.restart TO True

11. ELSE

12. SET self.game.restart TO True

13. END IF

14. END PROCEDURE

## Pseudocode for End User Requirements

End-User Requirement 1 & 2:

1. PROCEDURE Controls(self, pressedKey)

2. SET self.lastKeyPressed TO pressedKey

3. IF pressedKey == pg.K\_a THEN

4. CALL self.shape.Move('left')

5. ELSE IF pressedKey == pg.K\_d THEN

6. CALL self.shape.Move('right')

7. ELSE IF pressedKey == pg.K\_s THEN

8. SET self.speedUp TO True

9. ELSE IF pressedKey == pg.K\_w THEN

10. CALL self.shape.Rotate('left')

11. ELSE IF pressedKey == pg.K\_e THEN

12. CALL self.shape.Rotate('right')

13. END IF

14. END PROCEDURE

End-User Requirement 1 & 2(validation):

1. FUNCTION IsCollide(self, pos)

2. SET x, y TO INT(pos.x), INT(pos.y)

3. IF (0 <= x < FIELD\_W) AND (y < FIELD\_H) AND ((y < 0) OR (NOT self.shape.game.fieldArray[y][x]))

THEN

4. RETURN False

5. END IF

6. RETURN True

7. END FUNCTION

End-User Requirement 3:

1. PROCEDURE GetPlayerIdInput(self)

2. SET pressedKey TO self.game.app.pressedKey

3. IF (pressedKey != pg.K\_BACKSPACE) AND (LEN(self.player) < 3) AND (pressedKey != NONE) AND

(pressedKey >= 97 AND pressedKey <= 122) THEN

4. SET self.player TO self.player + CHR(pressedKey).UPPER()

5. ELSE IF pressedKey == pg.K\_BACKSPACE THEN

6. SET self.player TO self.player[:LEN(self.player) - 1]

7. ELSE IF (LEN(self.player) == 3) AND (pressedKey == pg.K\_RETURN) THEN

8. IF NOT(self.prevPlayerTop5 == 5) THEN

9. CALL self.game.app.mySQLServer.InsertStatement('INSERT INTO highscores(id, score)

VALUES("' + STR(self.player) + '", "' + STR(INT(self.game.score)) + '")')

10. SET self.game.restart TO True

11. ELSE

12. SET self.game.restart TO True

13. END IF

14. END PROCEDURE

End-User Requirement 4:

1. PROCEDURE DrawHighscores(self)

2. SET textHeight TO 0

3. SET scoreHeaderText TO CALL self.DrawText(HIGHSCORE\_HEADER\_OFFSET,

HIGHSCORE\_HEADER, textColor=TEXT\_COLOR, antialias=True)

4. FOR i, prevPlayerInfo IN ENUMERATE(self.prevPlayerTop5) DO

5. FOR text IN (self.prevPlayerTop5[0:i]) DO

6. textHeight += text.get\_height()

7. END FOR

8. CALL self.game.app.display.blit(prevPlayerInfo, (HIGHSCORE\_HEADER\_OFFSET + (0,

scoreHeaderText.get\_height()) + (0, textHeight)))

9. SET textHeight TO 0

10. END FOR

11.END PROCEDURE

## Design of Advanced Higher Concepts

I will implement a bubble sort algorithm to sort high scores for the leader board:

Pseudocode:

1. FUNCTION BubbleSort(arr, indexOfNum)

2. SET tempHolder TO 0

3. SET outer TO len(arr) - 1

4. SET noSwap TO False

5. WHILE (outer != -1 OR noSwap == False) DO

6. SET noSwap TO True

7. FOR inner IN RANGE(0, outer) DO

8. IF arr[inner][indexOfNum] < arr[inner + 1][indexOfNum] THEN

9. SET tempHolder TO arr[inner][indexOfNum]

10. arr[inner][indexOfNum] = arr[inner + 1][indexOfNum]

11. arr[inner + 1][indexOfNum] = tempHolder

12. SET noSwap TO False

13. END IF

14. END FOR

15. outer -= 1

16. END WHILE

17. RETURN arr

18. END FUNCTION

My program will also make use of an array of objects where the array holds objects of shapes that are used in the game. The pseudocode for this is shown below the Block UML Class diagram.

## UML Class Diagrams

|  |
| --- |
| app |
| Attributes |
| +display : Surface |
| +screen : Surface |
| +keyHeld : list |
| +clock : Clock |
| +game : Game |
| Methods |
| -\_\_init\_\_(app) |
| +Setup(app) |
| +SetupDatabaseServer(app) |
| +MakeMySQLServer(app) : MySQLServer |
| +SetTimer(app) |
| +CheckForEvents(app) |
| +Uppdate(app) |
| +Draw(app) |
| +Main(app) |

|  |
| --- |
| Game |
| Attributes |
| +app : app |
| +overlay : Surface |
| +blockGroup : Group |
| +fieldArray : list |
| +shape : Shape |
| +font : Font |
| +speedup : Boolean |
| +lines : int |
| +score : int |
| +gameOver : Boolean |
| +restart : Boolean |
| Methods |
| -\_\_init\_\_(Game, app) |
| +DrawGrid(Game) |
| +GetOverlay(Game, String) : Surface |
| +DrawOverlay(Game) |
| +GetFont(Game, String, int, Boolean, Boolean) : Font |
| +GetText(Game, String, Tuple, Boolean) : Surface |
| +GetFieldArray(Game) : list |
| +GetShape(Game) : Shape |
| +ClearLines(Game) : int |
| +UpdateScore(Game, int) |
| +DrawScore(Game) |
| +PutBlocksInFieldArray(Game) |
| +CheckIfShapeLanded(Game) |
| +Controls(Game, int) |
| +CheckIfGameOver(Game) |
| +Update(Game) |
| +Draw(Game) |

|  |
| --- |
| Shape |
| Attributes |
| +game : Game |
| +shape : String |
| +blocks : list |
| +landed : Boolean |
| Methods |
| -\_\_init\_\_(Shape, Game) |
| +GetShape(Shape) : String |
| +GetArrayOfBlocks(Shape) : list |
| +IsCollide(Shape, Vector2) : Boolean |
| +Move(Shape, String) |
| +Rotate(Shape, String) |
| +Update(Shape) |

|  |
| --- |
| Block |
| Attributes |
| +pos : Vector2 |
| +shape : Shape |
| +living : Boolean |
| +image : Surface |
| +rect : Rect |
| Methods |
| -\_\_init\_\_(Block, Shape, Vector2) |
| +IsAlive(Block) |
| +IsCollide(Block, Vector2) : Boolean |
| +SetRectPos(Block) |
| +Rotate(Block, Vector2, String) |
| +Update(Block) |
| +Draw(Block) |

## Design of Array of Objects

DECLARE blocks[4] AS INITIALLY ARRAY OF BLOCK

|  |
| --- |
| EndGameController |
| Attributes |
| +game : Game |
| +player : String |
| +font : Font |
| +prevPlayerTop5 : list |
| Methods |
| -\_\_init\_\_(EndGameController, Game) |
| +GetFont(EndGameController, String, int, Boolean, Boolean) : Font |
| +GetHighScoresToSurface(EndGameController) : list |
| +GetPlayerIdInput(EndGameController) |
| +GetText(EndGameController, String, Tuple, Boolean) : Surface |
| +DrawText(EndGameController, Tuple, String, Tuple, Boolean) : Surface |
| +DrawHighscores(EndGameController) |
| +DrawPlayerScore(EndGameController) |
| +Update(EndGameController) |
| +Draw(EndGameController) |

|  |
| --- |
| MySQLServer |
| Attributes |
| +host : String |
| +user : String |
| +password : String |
| +dbConnection : MySQLConnection |
| +cursor : MySQLCursor |
| Methods |
| -\_\_init\_\_(MySQLServer, String, String, String) |
| +ConnectToServer(MySQLServer) |
| +CreateDB(MySQLServer, String) : Boolean |
| +UseDB(MySQLServer, String) : MySQLCursor |
| +CreateTable(MySQLServer, String) : MySQLCursor |
| +SelectStatement(MySQLServer, String) : MySQLCursor |
| +InsertStatement (MySQLServer, String) : MySQLCursor |
| +UpdateStatement(MySQLServer, String) : MySQLCursor |
| +DeleteStatement (MySQLServer, String) : MySQLCursor |

## Design of integration

### Entity Relationship Diagram

Diagram

Description automatically generated

### Data Dictionary

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Constraint |
| id | Varchar(3) | NOT NULL |
| score | int | NOT NULL |

### Query Designs

#### Creates Table

|  |  |
| --- | --- |
| Create table | HighScores(  id varchar(3) NOT NULL,  score INT NOT NULL  ); |

#### Use Table

|  |  |
| --- | --- |
| Use Table | HighScores |

#### Gets Top 5 Players

|  |  |
| --- | --- |
| Fields | \* |
| Tables | highscores |
| Criteria |  |
| Grouping |  |
| Ordering | Score DESC |
| Limit | 5 |

#### Insert Player Score into database

|  |  |
| --- | --- |
| INSERT INTO | Highscores(id, score) |
| VALUES | (Player.id, player.score) |

## User-Interface

### Playing UI

Chart, histogram

Description automatically generated

### Game Over UI

Diagram

Description automatically generated

# Implementation

## AH Concepts Implementation

## Bubble Sort

Text

Description automatically generated

Text

Description automatically generated

## Array of Objects

Text

Description automatically generated with medium confidence

Text, letter

Description automatically generated

## Implemented Integration

Creates blockgame database:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Creates highscores table in blockgame database:

Text

Description automatically generated

Use Database:

Text

Description automatically generated

Inserts the players score into table:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generatedTable

Description automatically generated with low confidence

These screenshots show the code that is used to produce the database and table, as well as the queries for this database.

## Implemented UI

Text

Description automatically generated

When game is running:

Chart

Description automatically generated

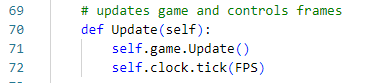
When game is over:

Text

Description automatically generated with medium confidence

## Implement Functional requirements

FR 1:



FR 2:

Text, letter

Description automatically generated

FR 3:

A picture containing text

Description automatically generated

FR 4:

A picture containing calendar

Description automatically generated

FR 5:

Text

Description automatically generated

FR 6:

Text

Description automatically generated

FR 7:

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Text, application

Description automatically generated

FR 8:

Text

Description automatically generated

FR 9:

Text

Description automatically generated

FR 10:

Graphical user interface, text, application, email

Description automatically generated

## Implemented End-User Requirements

End-User Requirement 1 & 2:

Text

Description automatically generated

End-User Requirement 1 & 2 (validation):

A picture containing text

Description automatically generated

End-User Requirement 3:

Graphical user interface, text, application, email

Description automatically generated

End-User Requirement 4:

Graphical user interface, application, Word

Description automatically generated with medium confidence

## New Skills/Knowledge

|  |  |  |
| --- | --- | --- |
| New skill/knowledge | Why was it needed for project | How it was applied to project |
| Pygame | To implement all the graphical features of the game, as well as all collisions | This was needed to create the UI and logic for the game |
| Rotation about an arbitrary point | To rotate coordinates around an arbitrary pivot point | This was used to implement the shape rotation |
| Python | To create the game and use Pygame | All the source code is written in Python using Pygame |
| MySQL Connector | To implement database integration | To create and query database and tables for highscores. |

### References

Pygame:

<https://www.pygame.org/docs/>

Rotation about an arbitrary point:

<https://www.ques10.com/p/22037/explain-the-steps-used-in-rotation-of-2d-object--1/#:~:text=Rotation%20about%20an%20Arbitrary%20Point,(See%20figure%201.).>

<https://stackoverflow.com/questions/2259476/rotating-a-point-about-another-point-2d>

Python:

<https://docs.python.org/3/>

MySQL Connector:

<https://dev.mysql.com/doc/connector-python/en/>

<https://www.w3schools.com/python/python_mysql_getstarted.asp>

## Log of ongoing Testing

|  |  |  |  |
| --- | --- | --- | --- |
| What is being tested | Issue Description | Description of Fix | References of how to solve |
| PutBlocksInFieldArray function | Position has a floating point error | Make block positions int by wrapping in int function | N/A |
| Collision on the X-coordinate | Shapes were able to move out width playable area | Changed the collision for X-coordinate to be < RATHER THAN <= | N/A |
| Player entering name after game is over | Player was able to enter only 2 letters rather then 3 | Changed the check length on the name entered to count one more character | N/A |
| Leaderboard after game ends | Leaderboard was rendered from lowest score to highest | Changed > to < in Bubble Sort | N/A |
| Database loading | Crashes if not running | Implement try except so program will inform user what is happening | N/A |

# Testing

## Plan for Final Testing

Testing of my solution will include beta testers that will play the game ensuring that all the functional and end-user requirements are met. To accomplish this, I will use unit testing to test core components of my program, to do this drivers and stubs will be used using comprehensive test data as well as debug print statements to query the state of variables.

Once unit tests have been completed Integrative testing will be used to ensure that all the modules work together.

After integrative testing system testing will begin and all modules will be tested and checked to ensure that they are all working as they should and that they work together.

Lastly acceptance testing will be done to ensure my solution is fit for purpose, meeting all the functional requirements.

## Evidence of Requirements Testing

### Screenshots Showing Inputs and Any Errors Generated for My Program

Shows inputs to rotate shape left:

Whiteboard

Description automatically generated with medium confidenceA picture containing chart

Description automatically generatedChart

Description automatically generated with medium confidenceA picture containing whiteboard

Description automatically generated

Shows inputs to rotate shape right:

A picture containing chart

Description automatically generated A picture containing chart

Description automatically generated

Shows inputs to move shape left:

A picture containing whiteboard

Description automatically generatedChart

Description automatically generated

Shows inputs to move shape to the right, also shows that the validation for movement works as the shape stays within the playable area:

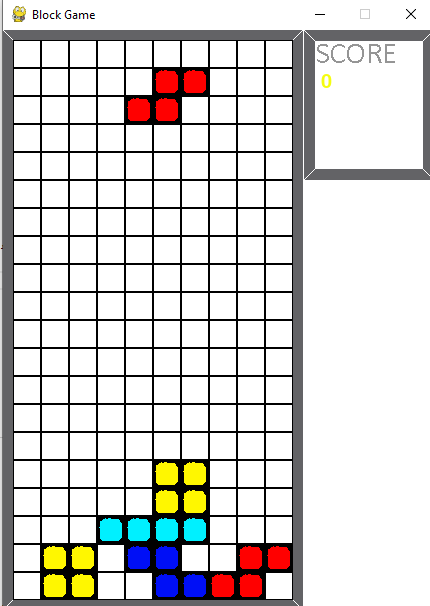
Chart

Description automatically generated with medium confidence Text, whiteboard

Description automatically generated

Shows the input to drop shape, also shows that blocks collide properly:

Text

Description automatically generated with medium confidence

### Screenshots Showing Successful Implementation of Advanced Higher Algorithms

Shows the bubble sort algorithm sorting leaderboard from highest to lowest scores:

Text

Description automatically generated with medium confidence Text

Description automatically generated with medium confidence Text

Description automatically generated with medium confidence

### Screenshots Showing Successful Implementation of SQL Queries

Shows player inputting name so score can be inserted into the database:

Text

Description automatically generated A picture containing application

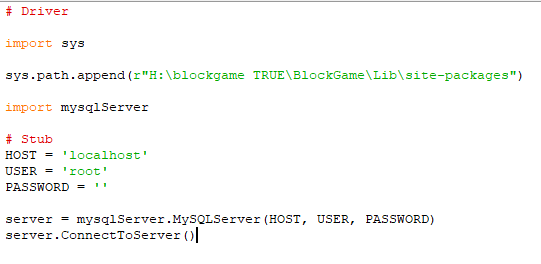
Description automatically generated Text

Description automatically generated

## Stub, Diver and Debug Print Statements Testing

### Testing database server connect function:

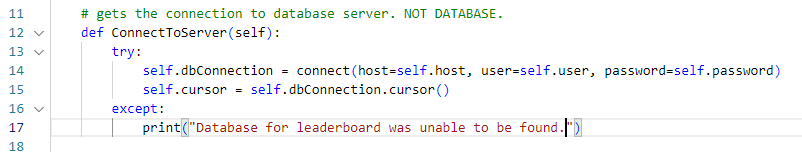
Driver and stub to test connecting to database server:



Error message:

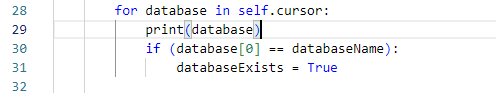


Fix:



### Print Debug Statements

Print statement used to determine whether the program was creating the database:



## Full System Testing

### By Programmer

This shows all the modules and components integrated together to create the entire game:

#### app, game, mysqlserver and blocks modules

Shows collision and movement testing:

Text, whiteboard

Description automatically generated A picture containing text

Description automatically generated

Shows score is updated correctly and that rotation works

Whiteboard

Description automatically generated Whiteboard

Description automatically generated

#### Endgame overlay

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated

### By Beta Testers

Beta testing was conducted using my classmates as the beta testers. Not only did this test the full system, but also gave valuable insight into my target end-users as all my classmates fell into the target end-user group. I also used this as acceptance testing for my game.

#### Results

Through beta testing the full system worked though there was a bug discovered. If you tried to start the application without first starting the database server the application would crash, without giving a concise error message. Even with the error though the overall review of the game was positive.

#### Screenshots of bugs discovered

Error:



Fix:



## Testing with persona and test cases

Persona 1:

Eustace Fargo is a 17-year-old S6 student . He doesn’t play computer games much. He is proficient at using a computer and will have no issues with accessibility and will be fine using a keyboard and mouse.

Persona 2:  
Wulf Sternhammer is a 70-year-old retiree. He doesn’t have much experience using electronics and has difficulty navigating complex applications on a computer.

Persona3:

Cassandra Anderson is a 45-year-old judge. She is proficient in using computers and often plays computer games, liking fast paced puzzle games most. She will have no problems adapting to Block Game and will have no accessibility problems.

## Results of Test Cases

Persona 1:

When completing persona testing with persona 1(main target for end user) they were able to play the game without having any troubles, quickly adapting to the UI. They had trouble with clearing lines and didn’t do great in their first few games only clearing an average of 7 lines. This average rose to 20 once they had played more.

Persona 2:

When completing persona testing with persona 2 they were able to start the game and get playing. They had some difficulty with keeping up with the blocks falling and with the UI, but as they played more, they kept improving. They had an average clearing of 4 lines on their first few attempts at the game. This then rose to 10 after playing more.

Persona 3:

When completing persona testing with persona 3 they easily started the game, quickly getting the hang of it. On their first few attempts at the game, they had cleared an average of 28 lines. This then rose to 40 once they played a bit more.

# Evaluation

## Evaluation of solution

### Fitness for Purpose

#### Matches Requirements Specification

My solution is fit for purpose as all the functional and end-user requirements are met as proven in the “Evidence for Requirements Testing” section. As well as this my solution also implements all the required advanced higher concepts as seen in the design and implementation section on top of this my project interfaces with a database by creating it and a table to store the data required by my program and using queries to grab the correct data to display the leaderboard in my program.

#### Results of Testing

System testing:

The result of my testing shows that all modules and subsystems work and that they work together as shown by my full system testing. My full system testing did show that if the database server wasn’t running the application would crash and that a try except block should be implemented to catch this error.

Persona testing:

The results of my persona testing shows that my program’s UI is understandable and that the controls and gameplay are intuitive.

## Evaluation of Solution in terms of:

### Efficiency

My solution is efficient as it makes use of algorithms such as Bubble sort to keep the efficiency high.

### Future Maintainability

My program is maintainable as it uses OOP concepts, such as encapsulation and inheritance, fully allowing easy understanding of the game loop and data flow. It uses objects, inheritance, and composition to allow easy maintainability, packing the variables with the functions that act on them. The entire program is very modular, so that each individual component of the program can be changed and tested easily.

### Robustness

My program is very robust as it filters out all exceptional inputs, only allowing the keys that it recognizes to be passed to the program. It also makes use of try except blocks to ensure that error are handled properly and that the user knows what is happening and how to handle an error if it occurs.

# Appendix

## Design of Modules for Block Game:

### App Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

import sys

sys.path.append\

    (r"H:\blockgame TRUE\BlockGame\Lib\site-packages")

import mysqlServer

import pygame as pg

from game import \*

from config import \*

from mySQLServerConfig import \*

# class to handle the running of the app

class app:

    # initialises app

    def \_\_init\_\_(self):

        pg.init()

        self.Setup()

        pg.display.set\_caption("Block Game")

        self.display = pg.display.set\_mode(DISPLAY\_SIZE)

        self.screen = pg.surface.Surface\

            ((FIELD\_W\*TILE\_SIZE, FIELD\_H\*TILE\_SIZE))

        self.SetTimer()

        self.keyHeld = [False, 0]

        self.clock = pg.time.Clock()

        self.game = Game(self)

    # setsup external game components

    def Setup(self):

        self.SetupDatabaseServer()

    def SetupDatabaseServer(self):

        self.mySQLServer = self.MakeMySQLServer()

        self.mySQLServer.ConnectToServer()

        databaseCreated = self.mySQLServer.CreateDB(BLOCKGAME\_DB)

        self.mySQLServer.UseDB(BLOCKGAME\_DB)

        if databaseCreated:

            self.mySQLServer.CreateTable(TABLES['highscores'])

    # makes mySQL server

    def MakeMySQLServer(self):

        return mysqlServer.MySQLServer\

            (self, SERVER['host'], SERVER['user'], SERVER['password'])

    # creates timmer for user events

    def SetTimer(self):

        self.userAnimEvent = pg.USEREVENT + 0

        self.userFastAnimEvent = pg.USEREVENT + 1

        self.animTrigger = False

        self.fastAnimTrigger = False

        pg.time.set\_timer(self.userAnimEvent, ANIM\_TIME)

        pg.time.set\_timer(self.userFastAnimEvent, FAST\_ANIM\_TIME)

    # checks for events in pygame

    def CheckForEvents(self):

        self.pressedKey = None

        self.animTrigger = False

        self.fastAnimTrigger = False

        for event in pg.event.get():

            if event.type == pg.QUIT:

                pg.quit()

                sys.exit()

            if (event.type == pg.KEYDOWN):

                 self.game.Controls(pressedKey=event.key)

                 self.pressedKey = event.key

            if event.type == self.userAnimEvent:

                self.animTrigger = True

            if event.type == self.userFastAnimEvent:

                self.fastAnimTrigger = True

    # updates game and controls frames

    def Update(self):

        self.game.Update()

        self.clock.tick(FPS)

    # draws updates to screen

    def Draw(self):

        self.screen.fill(FIELD\_COLOR)

        self.game.Draw()

        if not(self.game.gameOver):

            self.display.blit(self.screen, FIELD\_OFFSET + (0, 0))

        pg.display.flip()

    # main game loop

    def Main(self):

        run = True

        while run:

            if self.game.restart:

                self.game = Game(self)

            self.CheckForEvents()

            self.Update()

            self.Draw()

application = app() # instantiates class that runs the application

application.Main() # calls main game loop

### MysqlServer Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

from mysql.connector import connect

import sys

# class to handle database server

class MySQLServer:

    # initialises the class

    def \_\_init\_\_(self, app, host, user, password):

        self.app = app

        self.host = host

        self.user = user

        self.password = password

        self.ConnectToServer()

    # gets the connection to database server. NOT DATABASE.

    def ConnectToServer(self):

        try:

            self.dbConnection = connect\

                (host=self.host, user=self.user, password=self.password)

            self.cursor = self.dbConnection.cursor()

        except:

            print("Database for leaderboard was unable to be found.")

            sys.exit()

    # used to create new database in server

    #  and makes sure it doesn't already exist

    def CreateDB(self, databaseName):

        databaseExists = False

        self.cursor.execute("SHOW DATABASES")

        for database in self.cursor:

            print(database)

            if (database[0] == databaseName):

                databaseExists = True

        if not(databaseExists):

            self.cursor.execute\

                (f'CREATE DATABASE {databaseName}')

            return True

        return False

    # used to pick which database to use

    def UseDB(self, databaseName):

        self.cursor.execute(f'USE {databaseName}')

        return self.cursor

    # used to create tables

    def CreateTable(self, statement):

        self.cursor.execute(f'{statement}')

        return self.cursor

    # used to select from table

    def SelectStatement(self, statement):

        self.cursor.execute(f'{statement}')

        return self.cursor

    # useed to insert values into table

    def InsertStatement(self, statement):

        self.cursor.execute(f'{statement}')

        return self.cursor

    # used to update the table

    def UpdateStatement(self, statement):

        self.cursor.execute(f'{statement}')

        return self.cursor

    # used to delete records from the table

    def DeleteStatement(self, statement):

        self.cursor.execute(f'{statement}')

        return self.cursor

### MySQLServerConfig Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

# server connection info

HOST = 'localhost'

USER = 'root'

PASSWORD = ''

# dictionary for server info

SERVER = {

    'host':HOST,

    'user':USER,

    'password':PASSWORD

}

# datebase for blockgame

BLOCKGAME\_DB = 'blockgame'

TABLES = {}

TABLES['highscores'] =(

    'CREATE TABLE HighScores('

    '   id varchar(3) NOT NULL,'

    '   score INT NOT NULL)'

)

# blockgame queries

BLOCK\_GAME\_QUERIES = {}

BLOCK\_GAME\_QUERIES['EndGameTop5'] = ('SELECT \* FROM highscores ORDER BY score DESC LIMIT 5')

### Game Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

from mySQLServerConfig import \*

from endgamecontroller import \*

import pygame as pg

from blocks import \*

from config import \*

class Game:

    # initialises game

    def \_\_init\_\_(self, app):

        self.app = app

        self.overlay = self.GetOverlay('Light')

        self.blockGroup = pg.sprite.Group()

        self.fieldArray = self.GetFieldArray()

        self.shape = self.GetShape()

        self.font = self.GetFont()

        self.speedUp = False

        self.lines = 0

        self.score = 0

        self.gameOver = False

        self.restart = False

    # draws play area grid

    def DrawGrid(self):

        for y in range(FIELD\_H):

            for x in range(FIELD\_W):

                pg.draw.rect(self.app.screen,

                             (0, 0, 0),

                               (x\*TILE\_SIZE,

                                 y\*TILE\_SIZE,

                                   TILE\_SIZE,

                                     TILE\_SIZE), 1)

    # loads the overlay for outside play area

    def GetOverlay(self, type='Light'):

        return pg.image.load\

            (f"{LOCAL\_DIR\_PATH}\\assets\\" +\

             f"overlay\\staticScreen{type}.png")

    # draws overlay to display

    def DrawOverlay(self):

        self.app.display.blit(self.overlay, (0, 0))

    # gets the font used in-game

    def GetFont(self, font=GAME\_FONT, \

                size=FONT\_SIZE, bold=False, italic=False):

        return pg.font.SysFont(font, size, bold, italic)

    # draws the score to the score box

    def GetText(self, text, color, antialias=False):

        textSurface = self.font.render\

            (text, antialias, color)

        return textSurface

    # gets the array with the state of each tile in play area

    def GetFieldArray(self):

        return [[0 for x in range(FIELD\_W)]\

                 for y in range(FIELD\_H)]

    # instantiates a new shape object

    def GetShape(self):

        return Shape(self)

    # checks and clears whole lines

    def ClearLines(self):

        lines = 0

        row = FIELD\_H - 1

        for y in range(FIELD\_H -1, -1, -1):

            for x in range(FIELD\_W):

                self.fieldArray[row][x] =\

                      self.fieldArray[y][x]

                if self.fieldArray[y][x]:

                    self.fieldArray[row][x].pos =\

                          pg.math.Vector2(x, row)

            if sum(map(bool, self.fieldArray[y])) < FIELD\_W:

                row -= 1

            else:

                for x in range(FIELD\_W):

                    lines += 1

                    self.fieldArray[row][x].living = False

                    self.fieldArray[row][x] = 0

        return lines

    # updates the score

    def UpdateScore(self, clearedLines):

        self.score += (SCORE\_BASE) \* (clearedLines\*\*SCORE\_MULT)

    # draws score to display

    def DrawScore(self):

        score = self.GetText\

            (str(self.score), TEXT\_COLOR, antialias=True) # get score surface

        self.app.display.blit(score, SCORE\_OFFSET) # show score

    # takes the position of the current shape

    # and updates field array tile states accordingly

    def PutBlocksInFieldArray(self):

        for block in self.shape.blocks:

            x, y = int(block.pos.x), int(block.pos.y)

            self.fieldArray[y][x] = block

    # checks if the current shape has landed and

    #  if true cleans up everything for new shape to appear

    def CheckIfShapeLanded(self):

        if self.shape.landed:

            self.CheckIfGameOver()

            self.speedUp = False

            self.PutBlocksInFieldArray()

            self.shape = self.GetShape()

    # controls what happens when a key is pressed

    def Controls(self, pressedKey):

        self.lastKeyPressed = pressedKey

        if pressedKey == pg.K\_a:

            self.shape.Move('left')

        elif pressedKey == pg.K\_d:

            self.shape.Move('right')

        elif pressedKey == pg.K\_s:

            self.speedUp = True

        elif pressedKey == pg.K\_w:

            self.shape.Rotate('left')

        elif pressedKey == pg.K\_e:

            self.shape.Rotate('right')

    # checks whether the game is over

    def CheckIfGameOver(self):

        for block in self.shape.blocks:

            if block.pos[1] \* TILE\_SIZE +\

                  FIELD\_OFFSET[1] <= FIELD\_OFFSET[1]:

                self.gameOver = True

    # updates the game state

    def Update(self):

        trigger = [self.app.animTrigger,\

                    self.app.fastAnimTrigger][self.speedUp]

        if not(self.gameOver):

            if trigger:

                self.lines = self.ClearLines()

                self.UpdateScore(self.lines)

                self.shape.Update()

                self.CheckIfShapeLanded()

                if self.gameOver:

                    self.endGameController =\

                          EndGameController(self)

            self.blockGroup.update()

        else:

            self.endGameController.Update()

    # draws the updates to the game to the screen

    def Draw(self):

        if not(self.gameOver):

            self.DrawOverlay()

            self.DrawGrid()

            self.blockGroup.draw(self.app.screen)

            self.DrawScore()

        else:

            self.endGameController.Draw()

### endgamecontroller Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

from config import \*

from mySQLServerConfig import \*

import mysqlServer

# sort algorithm to sort highscores from highest to lowest

def BubbleSort(arr, indexOfNum):

    tempHolder = 0

    outer = len(arr) - 1

    noSwap = False

    while (outer != -1 or noSwap == False):

        noSwap = True

        for inner in range(0, outer):

            if arr[inner][indexOfNum] < arr[inner + 1][indexOfNum]:

                tempHolder = arr[inner][indexOfNum]

                arr[inner][indexOfNum] = arr[inner + 1][indexOfNum]

                arr[inner + 1][indexOfNum] = tempHolder

                noSwap = False

        outer -= 1

    return arr

# class to control endgame display

class EndGameController():

    def \_\_init\_\_(self, game):

        self.game = game

        self.player = ''

        self.font = self.GetFont()

        self.prevPlayerTop5 = self.GetHighScoresToSurface()

    # gets font to use at endgame

    def GetFont(self, font=GAME\_FONT, size=ENDGAME\_SIZE,

                bold=False, italic=False):

        return pg.font.SysFont(font, size, bold, italic)

    # gets highscores

    def GetHighScoresToSurface(self):

        highscoresSurfaces = []

        highscores = self.game.app.mySQLServer.SelectStatement\

            (BLOCK\_GAME\_QUERIES['EndGameTop5'])

        highscores = highscores.fetchall()

        for i in range(0, len(highscores)):

            highscores[i] = [highscores[i][0], highscores[i][1]]

        if len(highscores) > 0:

            highscores = BubbleSort(highscores, 1)

        for (id, score) in highscores:

            prevPlayerText = str(id) + ' ' + str(score)

            prevPlayerTextSurface = self.GetText\

                (prevPlayerText, TEXT\_COLOR, antialias=True)

            highscoresSurfaces.append(prevPlayerTextSurface)

        return highscoresSurfaces

    # gets the players inputs for endgame overlay

    def GetPlayerIdInput(self):

        pressedKey = self.game.app.pressedKey

        if (pressedKey != pg.K\_BACKSPACE) and (len(self.player) < 3) \

            and (pressedKey != None) and \

                (pressedKey >= 97 and pressedKey <= 122):

            self.player = self.player + chr(pressedKey).upper()

        elif pressedKey == pg.K\_BACKSPACE:

            self.player = self.player[:len(self.player) - 1]

        elif (len(self.player) == 3) and (pressedKey == pg.K\_RETURN):

            if not(self.prevPlayerTop5 == 5):

                self.game.app.mySQLServer.InsertStatement\

                    (f'INSERT INTO highscores(id, score) VALUES' +\

                     f'("{str(self.player)}", "{int(self.game.score)}")')

                self.game.restart = True

            else:

                self.game.restart = True

    # gets text using font

    def GetText(self, text, textColor=TEXT\_COLOR, antialias=False):

        return self.font.render(text, antialias, textColor)

    # draws text to screen

    def DrawText(self, pos, text, textColor=TEXT\_COLOR, antialias=False):

        Text = self.font.render(text, antialias, textColor)

        self.game.app.display.blit(Text, pos)

        return Text

    # draws scores to display

    def DrawHighscores(self):

        textHeight = 0

        scoreHeaderText = self.DrawText\

            (HIGHSCORE\_HEADER\_OFFSET,

             HIGHSCORE\_HEADER, textColor=TEXT\_COLOR, antialias=True)

        for i, prevPlayerInfo in enumerate(self.prevPlayerTop5):

            for text in (self.prevPlayerTop5[0:i]):

                textHeight += text.get\_height()

            self.game.app.display.blit\

                (prevPlayerInfo,

                 (HIGHSCORE\_HEADER\_OFFSET + \

                  (0, scoreHeaderText.get\_height()) + (0, textHeight)))

            textHeight = 0

    # draws the players score to the screen

    def DrawPlayerScore(self):

        textHeight = 0

        playerHeader = self.DrawText\

            (PLAYER\_SCORE\_HEADER\_OFFSET,

             PLAYER\_HEADER, textColor=TEXT\_COLOR, antialias=True)

        player = self.GetText(self.player, TEXT\_COLOR, antialias=True)

        score = self.GetText(str(self.game.score), TEXT\_COLOR, antialias=True)

        playerInfoTup = (player, score)

        for i, playerInfo in enumerate(playerInfoTup):

            for text in (playerInfoTup[0:i]):

                textHeight += text.get\_height()

            self.game.app.display.blit\

                (playerInfo,

                 (PLAYER\_SCORE\_HEADER\_OFFSET + \

                  (0, playerHeader.get\_height()) + (0, textHeight)))

            textHeight = 0

    # Updates end game screen

    def Update(self):

        self.GetPlayerIdInput()

    # draws end game screen

    def Draw(self):

        self.game.app.display.fill((0, 0, 0))

        self.game.app.screen.fill((0, 0, 0))

        self.DrawHighscores()

        self.DrawPlayerScore()

Blocks Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

import pygame as pg

from config import \*

import random

# class to define the block components of shape

class Block(pg.sprite.Sprite):

    # initialises block object

    def \_\_init\_\_(self, shape, pos):

        self.pos = pg.math.Vector2(pos) + INIT\_START\_POS

        self.shape = shape

        self.living = True

        self.image = pg.image.load\

            (f"{LOCAL\_DIR\_PATH}\\assets\\blocks\\" +\

             f"{self.shape.shape}.png").convert()

        self.rect = pg.rect.Rect\

            (self.pos[0]\*TILE\_SIZE,

             self.pos[1]\*TILE\_SIZE, TILE\_SIZE, TILE\_SIZE)

        super().\_\_init\_\_(shape.game.blockGroup)

    # if block is alive(whether it has been clearedd)

    def IsAlive(self):

        if not(self.living):

            self.kill()

    # gets if the block has collided with something it can't pass

    def IsCollide(self, pos):

        x, y = int(pos.x), int(pos.y)

        if (0 <= x < FIELD\_W) and (y < FIELD\_H) and \

            ((y < 0) or \

             (not self.shape.game.fieldArray[y][x])):

            return False

        return True

    # sets position and size of rect

    def SetRectPos(self):

        self.rect.topleft = self.pos \* TILE\_SIZE

    # rotates blocks around pivot point

    def Rotate(self, pivotPos, direction):

        if not(self.shape.shape == 'O'):

            if direction=='left':

                diffPos = pg.math.Vector2\

                    (self.pos - pivotPos)

                rotatedPos = diffPos.rotate(-90)

                rotatedPos += pivotPos

                return rotatedPos

            elif direction=='right':

                diffPos = pg.math.Vector2\

                    (self.pos - pivotPos)

                rotatedPos = diffPos.rotate(90)

                rotatedPos += pivotPos

                return rotatedPos

        else:

            return self.pos

    # updates block

    def update(self):

        self.IsAlive()

        self.SetRectPos()

    # draws block to screen

    def Draw(self):

        self.shape.game.app.screen.blit\

            (self.image, self.pos\*TILE\_SIZE)

class Shape:

    # initialises shape object

    def \_\_init\_\_(self, game):

        self.game = game

        self.shape = self.GetShape()

        self.blocks = self.GetArrayOfBlocks()

        self.landed = False

    # chooses a shape

    def GetShape(self):

        return random.choice(list(SHAPES.keys()))

    # gets array of blocks that make up the shape

    def GetArrayOfBlocks(self):

        return \

            [Block(self, pos)\

              for pos in SHAPES[self.shape]]

    # checks if any blocks in shape collide

    def IsCollide(self, newBlockPos):

        return any\

            (map(Block.IsCollide, self.blocks, newBlockPos))

    # moves all blocks in the shape

    def Move(self, direction):

        moveDirection = MOVEMENT\_DIR[direction]

        # creates array of blocks new positions

        newBlockPos = [block.pos + moveDirection\

                        for block in self.blocks]

        # checks if any of the blocks new positions collide with anything

        isCollide = self.IsCollide(newBlockPos)

        # if no blocks collide

        if not (isCollide) and not (self.landed):

            for block in self.blocks:

                block.pos += moveDirection

        elif direction=='down':

            self.landed = True

    # rotates the current block and makes sure nothing collides

    def Rotate(self, direction):

        pivotPos = self.blocks[0].pos

        newBlockPos = [block.Rotate(pivotPos, direction)\

                        for block in self.blocks]

        isCollide = self.IsCollide(newBlockPos)

        if not(isCollide):

            for i, block in enumerate(self.blocks):

                block.pos = newBlockPos[i]

    # updates shape

    def Update(self):

        self.Move('down')

### Config Module:

# Adam Kennedy

# 16/03/2023

# Advanced Higher Project

import pygame as pg

import os

# dirs

LOCAL\_DIR\_PATH = os.path.dirname(os.path.realpath(\_\_file\_\_))

SCORE\_DIR = os.path.join(LOCAL\_DIR\_PATH, 'highscore')

# display variables

TILE\_SIZE = 28

FIELD\_W = 10

FIELD\_H = 20

FIELD\_OFFSET = pg.math.Vector2(10, 10)

SCORE\_OFFSET = pg.math.Vector2(318, 42)

DISPLAY\_SIZE = (431, 580)

FPS = 60

FIELD\_COLOR = (255, 255, 255)

SCORE\_BASE = 10

SCORE\_MULT = 2

PLAY\_AREA\_GRID\_COLOR = (255, 255, 255)

# end game high score pos

HIGHSCORE\_HEADER = 'TOP 5'

HIGHSCORE\_HEADER\_OFFSET = pg.math.Vector2(20, 50)

PLAYER\_HEADER = 'YOUR SCORE'

PLAYER\_SCORE\_HEADER\_OFFSET = pg.math.Vector2(230, 50)

# game start init

INIT\_START\_POS = (5, 0)

# text

GAME\_FONT = 'hellvetica'

FONT\_SIZE = 30

ENDGAME\_SIZE = 34

TEXT\_COLOR = (244, 252, 17)

# shapes

SHAPES = {

    'T':[(0, 0), (-1, 0), (1, 0), (0, 1)],

    'I':[(0, 0), (-1, 0), (-2, 0), (1, 0)],

    'S':[(0, 0), (-1, 0), (0, -1), (1, -1)],

    'Z':[(0, 0), (1, 0), (0, -1), (-1, -1)],

    'L':[(0, 0), (-1, 0), (-1, 1), (1, 0)],

    'J':[(0, 0), (-1, 0), (1, 0), (-1, -1)],

    'O':[(0, 0), (1, 0), (0, -1), (1, -1)]

}

# movement

MOVEMENT\_DIR = {

    'left':(-1, 0),

    'right':(1, 0),

    'down':(0, 1)

}

# animation trigger

ANIM\_TIME = 240

FAST\_ANIM\_TIME = 24