Snake Evolution

Generated by Doxygen 1.9.6

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Package org.app

Classes

- class App

  Class to launch the game.

5.2 Package org.engine

Classes

- class GameFrame

  The GameFrame class is a part of the Game Engine system that handles switching and displaying appropriate game states.
- enum GameState

  Represents all possible game states.
- interface StateChangeListener

  Interface to allow state switching in the engine from the states (Observer).

5.3 Package org.objects

Packages

- package food
- package obstacle

Classes

- class Snake

  Represents a snake object.

5.4 Package org.objects.food

Classes

- class BonusFood

  Represents bonus food.
- class Food

  Represents a food item.
- enum FoodType

  Represents all possible food types.

5.5 Package org.objects.obstacle

Classes

- class Obstacle

  Represents an obstacle.
- class ObstacleList

  Represents a List of all existing obstacles in the game.

5.6 Package org.panels

Classes

- class BgPanel

  A Panel that represents the game's background to be reused in every game screen.
- class GameOver

  A panel that represents the game-over screen.
- class GamePanel

  Represents the gameplay state.
- class Leaderboard

  Panel representing the Leaderboard screen.
- class MainMenu

  A panel representing the main menu.
- class Tutorial

  Represents a tutorial screen.

5.7 Package org.utilities

Classes

- class CellPosition

  Represents the position on screen in cell-system.
- enum Direction

  Describes all possible directions.
- class GameButton

  A JButton that conforms to the specified design.
- interface GameConstants

  Defines constants used in the game.
- class Player

  Represents a player that can be added to the leaderboard.

Chapter 6

Class Documentation

6.1 org.app.App Class Reference

Class to launch the game.

Static Public Member Functions

- static void main (String[ ] args)

6.1.1 Detailed Description

Class to launch the game.

Definition at line 6 of file App.java.

6.1.2 Member Function Documentation

6.1.2.1 main()

```
static void org.app.App.main (
            String[ ] args )  [static]
```

Definition at line 7 of file App.java.

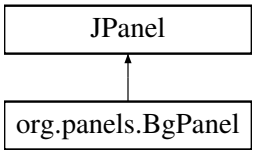
The documentation for this class was generated from the following file:

- App.java

6.2 org.panels.BgPanel Class Reference

A Panel that represents the game's background to be reused in every game screen.

Inheritance diagram for org.panels.BgPanel:



Public Member Functions

- BgPanel ()

  Constructs a JPanel matching the screen's size with a specified background color.
- void paintComponent (Graphics g)

  Overrides JPanel.paintComponent() to allow adding and drawing the panel to the GameFrame.

6.2.1 Detailed Description

A Panel that represents the game's background to be reused in every game screen.

Extends JPanel.

Author

Maksims Orlovs

Definition at line 12 of file BgPanel.java.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 BgPanel()

```
org.panels.BgPanel.BgPanel ( )
```

Constructs a JPanel matching the screen's size with a specified background color.

Author

Maksims Orlovs

Definition at line 17 of file BgPanel.java.

6.2.3 Member Function Documentation

6.2.3.1 paintComponent()

```
void org.panels.BgPanel.paintComponent (
            Graphics g )
```

Overrides JPanel.paintComponent() to allow adding and drawing the panel to the GameFrame.

Draws the background and the borders on the screen, specified by the design.

Parameters

g	graphics component supplied by the GameFrame

Author

Maksims Orlovs

Victoria Rönnlid (co-author)

Definition at line 32 of file BgPanel.java.

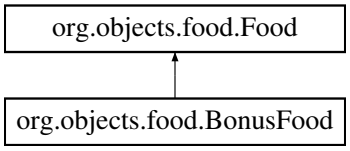
The documentation for this class was generated from the following file:

- BgPanel.java

6.3 org.objects.food.BonusFood Class Reference

Represents bonus food.

Inheritance diagram for org.objects.food.BonusFood:



Public Member Functions

- BonusFood ()

  Creates a BonusFood instance.
- void respawn ()

  Overrides default food respawning.
- void draw (Graphics2D frame)

  Method to draw the food item onto the screen (frame).

Public Member Functions inherited from org.objects.food.Food

- Food ()

  Creates a food item of default type in a random playable cell.
- void respawn ()

  Sets the position to a random playable cell.
- void draw (Graphics2D frame)

  Draws the food object onto the supplied frame in its current position.
- CellPosition getFoodLocation ()

  Getter for the current food position.
- FoodType getFoodType ()

  Getter for the food type.

Private Member Functions

- void randType ()

  Helper method to generate and assign a random type.

Private Attributes

- String icon

Additional Inherited Members

Protected Attributes inherited from org.objects.food.Food

- CellPosition foodLocation
- Random rand
- Color color
- FoodType type

Static Protected Attributes inherited from org.objects.food.Food

- static final int BORDER_SIZE = 2

6.3.1 Detailed Description

Represents bonus food.

Extends food. Includes random type selection and appropriate icon and color selection.

Author

Maksims Orlovs

Fatemeh Akbarifar (co-author)

Definition at line 13 of file BonusFood.java.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 BonusFood()

```
org.objects.food.BonusFood.BonusFood ( )
```

Creates a BonusFood instance.

Same as food, but uses overridden respawn() that includes random type generation.

See also

Food::Food()

Author

Maksims Orlovs

Definition at line 21 of file BonusFood.java.

6.3.3 Member Function Documentation

6.3.3.1 draw()

```
void org.objects.food.BonusFood.draw (
            Graphics2D frame )
```

Method to draw the food item onto the screen (frame).

Same as Food, but includes an appropriate icon based on the type.

Parameters

frame	Swing Graphics2D object that represents the current frame to be updated.

See also

Food::draw(Graphics2D)

Author

Maksims Orlovs

Reimplemented from org.objects.food.Food.

Definition at line 44 of file BonusFood.java.

6.3.3.2 randType()

```
void org.objects.food.BonusFood.randType ( )  [private]
```

Helper method to generate and assign a random type.

Assigns the color and icon according to the type.

See also

FoodType

Author

Fatemeh Akbarifar
Maksims Orlovs (co-author)

Definition at line 65 of file BonusFood.java.

6.3.3.3 respawn()

```
void org.objects.food.BonusFood.respawn ( )
```

Overrides default food respawning.

Same as Food, but includes random type generation.

See also

Food::respawn()

Author

Maksims Orlovs

Reimplemented from org.objects.food.Food.

Definition at line 31 of file BonusFood.java.

6.3.4 Member Data Documentation

6.3.4.1 icon

```
String org.objects.food.BonusFood.icon  [private]
```

Definition at line 14 of file BonusFood.java.

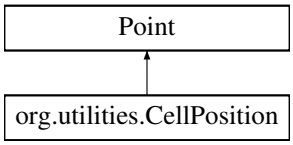
The documentation for this class was generated from the following file:

- BonusFood.java

6.4 org.utilities.CellPosition Class Reference

Represents the position on screen in cell-system.

Inheritance diagram for org.utilities.CellPosition:



Public Member Functions

- CellPosition ()
  Creates the CellPosition object with default cell 0,0.
- CellPosition (int initCellX, int initCellY)
  Creates the CellPosition with supplied indexes.
- Point getCoordinates ()
  Converts the cell indexes into top-left pixel coordinates of the cell.
- boolean equals (Object object)
  Compares the CellPosition to another object.

6.4.1 Detailed Description

Represents the position on screen in cell-system.

Stores x and y index of the cell.

Author

Maksims Orlovs

Definition at line 10 of file CellPosition.java.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 CellPosition() [1/2]

```
org.utilities.CellPosition.CellPosition ( )
```

Creates the CellPosition object with default cell 0,0.

Author

Maksims Orlovs

Definition at line 16 of file CellPosition.java.

6.4.2.2 CellPosition() [2/2]

```
org.utilities.CellPosition.CellPosition (
            int initCellX,
            int initCellY )
```

Creates the CellPosition with supplied indexes.

Parameters

initCellX	cell x index
initCellY	cell y index

Author

Maksims Orlovs

Definition at line 26 of file CellPosition.java.

6.4.3 Member Function Documentation

6.4.3.1 equals()

```
boolean org.utilities.CellPosition.equals (
            Object object )
```

Compares the CellPosition to another object.

Parameters

object	an object to be compared with this object

Returns

true if x and y indexes of both objects are equal

Author

Maksims Orlovs

Definition at line 47 of file CellPosition.java.

6.4.3.2 getCoordinates()

```
Point org.utilities.CellPosition.getCoordinates ( )
```

Converts the cell indexes into top-left pixel coordinates of the cell.

Returns

Point representing the position of the top-left corner of the cell in pixels.

Author

Maksims Orlovs

Definition at line 36 of file CellPosition.java.

The documentation for this class was generated from the following file:

- CellPosition.java

6.5 org.utilities.Direction Enum Reference

Describes all possible directions.

Public Attributes

- UP
- DOWN
- LEFT
- RIGHT

6.5.1 Detailed Description

Describes all possible directions.

Author

Maksims Orlovs

Definition at line 7 of file Direction.java.

6.5.2 Member Data Documentation

6.5.2.1 DOWN

`org.utilities.Direction.DOWN`

Definition at line 8 of file Direction.java.

6.5.2.2 LEFT

`org.utilities.Direction.LEFT`

Definition at line 8 of file Direction.java.

6.5.2.3 RIGHT

`org.utilities.Direction.RIGHT`

Definition at line 8 of file Direction.java.

6.5.2.4 UP

`org.utilities.Direction.UP`

Definition at line 8 of file Direction.java.

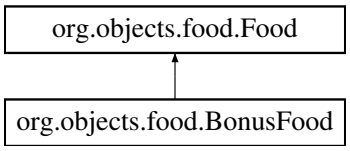
The documentation for this enum was generated from the following file:

- Direction.java

6.6 org.objects.food.Food Class Reference

Represents a food item.

Inheritance diagram for org.objects.food.Food:



Public Member Functions

- Food ()

  Creates a food item of default type in a random playable cell.
- void respawn ()

  Sets the position to a random playable cell.
- void draw (Graphics2D frame)

  Draws the food object onto the supplied frame in its current position.
- CellPosition getFoodLocation ()

  Getter for the current food position.
- FoodType getFoodType ()

  Getter for the food type.

Protected Attributes

- CellPosition foodLocation
- Random rand
- Color color
- FoodType type

Static Protected Attributes

- static final int BORDER_SIZE = 2

6.6.1 Detailed Description

Represents a food item.

Author

Fatemeh Akbarifar

Definition at line 14 of file Food.java.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Food()

```
org.objects.food.Food.Food ( )
```

Creates a food item of default type in a random playable cell.

Author

Fatemeh Akbarifar

Maksims Orlovs (co-author)

Definition at line 27 of file Food.java.

6.6.3 Member Function Documentation

6.6.3.1 draw()

```
void org.objects.food.Food.draw (
            Graphics2D frame )
```

Draws the food object onto the supplied frame in its current position.

Parameters

frame	Swing Graphics2D object that represents the current frame to be updated

Author

Fatemeh Akbarifar

Reimplemented in org.objects.food.BonusFood.

Definition at line 50 of file Food.java.

6.6.3.2 getFoodLocation()

```
CellPosition org.objects.food.Food.getFoodLocation ( )
```

Getter for the current food position.

Returns

CellPosition representing the current position

Author

Fatemeh Akbarifar

Definition at line 73 of file Food.java.

6.6.3.3 getFoodType()

```
FoodType org.objects.food.Food.getFoodType ( )
```

Getter for the food type.

Returns

FoodType of the food item

Author

Fatemeh Akbarifar

Definition at line 82 of file Food.java.

6.6.3.4 respawn()

```
void org.objects.food.Food.respawn ( )
```

Sets the position to a random playable cell.

Author

Fatemeh Akbarifar

Reimplemented in org.objects.food.BonusFood.

Definition at line 39 of file Food.java.

6.6.4 Member Data Documentation

6.6.4.1 BORDER_SIZE

```
final int org.objects.food.Food.BORDER_SIZE = 2  [static], [protected]
```

Definition at line 15 of file Food.java.

6.6.4.2 color

```
Color org.objects.food.Food.color  [protected]
```

Definition at line 19 of file Food.java.

6.6.4.3 foodLocation

```
CellPosition org.objects.food.Food.foodLocation  [protected]
```

Definition at line 17 of file Food.java.

6.6.4.4 rand

```
Random org.objects.food.Food.rand  [protected]
```

Definition at line 18 of file Food.java.

6.6.4.5 type

```
FoodType org.objects.food.Food.type  [protected]
```

Definition at line 20 of file Food.java.

The documentation for this class was generated from the following file:

- Food.java

6.7 org.objects.food.FoodType Enum Reference

Represents all possible food types.

Public Attributes

- DEFAULT
- SPEEDFOOD
- SLOWFOOD
- MINUSFOOD
- PLUSFOOD
- CONTROLINVERTER

6.7.1 Detailed Description

Represents all possible food types.

Author

Fatemeh Akbarifar

Definition at line 7 of file FoodType.java.

6.7.2 Member Data Documentation

6.7.2.1 CONTROLINVERTER

`org.objects.food.FoodType.CONTROLINVERTER`

Definition at line 9 of file FoodType.java.

6.7.2.2 DEFAULT

`org.objects.food.FoodType.DEFAULT`

Definition at line 8 of file FoodType.java.

6.7.2.3 MINUSFOOD

`org.objects.food.FoodType.MINUSFOOD`

Definition at line 8 of file FoodType.java.

6.7.2.4 PLUSFOOD

`org.objects.food.FoodType.PLUSFOOD`

Definition at line 8 of file FoodType.java.

6.7.2.5 SLOWFOOD

`org.objects.food.FoodType.SLOWFOOD`

Definition at line 8 of file FoodType.java.

6.7.2.6 SPEEDFOOD

`org.objects.food.FoodType.SPEEDFOOD`

Definition at line 8 of file FoodType.java.

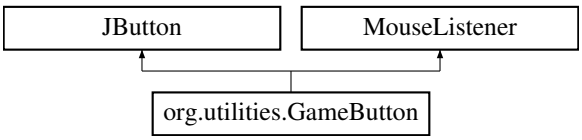
The documentation for this enum was generated from the following file:

- FoodType.java

6.8 org.utilities.GameButton Class Reference

A JButton that conforms to the specified design.

Inheritance diagram for org.utilities.GameButton:



Public Member Functions

- GameButton (String standardText)

  Creates and sets up a button object and applies the specified style to it.
- void editButton (GameButton button)

  Applies the design to the given button.
- void onHover (boolean hovering)

  Adds and removes a selection effect to the button by adding special characters to the button text.
- void mouseClicked (MouseEvent e)
- void mousePressed (MouseEvent e)
- void mouseReleased (MouseEvent e)
- void mouseEntered (MouseEvent e)

  Applies the hovering effect when mouse enters the button area.
- void mouseExited (MouseEvent e)

  Removes the hovering effect when mouse exits the button area.

Private Attributes

- final String standardText

6.8.1 Detailed Description

A JButton that conforms to the specified design.

Author

Victoria Rönnlid

Definition at line 12 of file GameButton.java.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 GameButton()

```
org.utilities.GameButton.GameButton (
            String standardText )
```

Creates and sets up a button object and applies the specified style to it.

Parameters

standardText	button text to display

Author

Victoria Rönnlid

Definition at line 20 of file GameButton.java.

6.8.3 Member Function Documentation

6.8.3.1 editButton()

```
void org.utilities.GameButton.editButton (
            GameButton button )
```

Applies the design to the given button.

Parameters

button	button to apply the design to

Author

Victoria Rönnlid

Definition at line 33 of file GameButton.java.

6.8.3.2 mouseClicked()

```
void org.utilities.GameButton.mouseClicked (
            MouseEvent e )
```

Definition at line 62 of file GameButton.java.

6.8.3.3 mouseEntered()

```
void org.utilities.GameButton.mouseEntered (
            MouseEvent e )
```

Applies the hovering effect when mouse enters the button area.

Parameters

e	the event to be processed

Author

Victoria Rönnlid

Definition at line 76 of file GameButton.java.

6.8.3.4 mouseExited()

```
void org.utilities.GameButton.mouseExited (
            MouseEvent e )
```

Removes the hovering effect when mouse exits the button area.

Parameters

e	the event to be processed

Author

Victoria Rönnlid

Definition at line 86 of file GameButton.java.

6.8.3.5 mousePressed()

```
void org.utilities.GameButton.mousePressed (
            MouseEvent e )
```

Definition at line 65 of file GameButton.java.

6.8.3.6 mouseReleased()

```
void org.utilities.GameButton.mouseReleased (
            MouseEvent e )
```

Definition at line 68 of file GameButton.java.

6.8.3.7 onHover()

```
void org.utilities.GameButton.onHover (
            boolean hovering )
```

Adds and removes a selection effect to the button by adding special characters to the button text.

Parameters

hovering	a flag that determines if the effect should be applied or removed (true - mouse is over the button, false otherwise)

Author

Victoria Rönnlid

Definition at line 51 of file GameButton.java.

6.8.4 Member Data Documentation

6.8.4.1 standardText

```
final String org.utilities.GameButton.standardText  [private]
```

Definition at line 13 of file GameButton.java.

The documentation for this class was generated from the following file:

- GameButton.java

6.9 org.utilities.GameConstants Interface Reference

Defines constants used in the game.

Static Public Attributes

- static final Point WINDOW_SIZE = new Point(800, 800)

  Point representing the window dimensions.
- static final int FPS = 60

  The base FPS of the game.
- static final int CELL_COUNT = 40

  The amount of cells in one row/column.
- static final int CELL_SIZE = WINDOW_SIZE.x / CELL_COUNT

  Size of one cell.
- static final int EFFECT_DURATION = 8000

  Duration of timed bonus food effects (in ms).
- static final int BORDER_THC = 5

  Thickness of the borders in pixels.
- static final int MARGIN_CELLS = 3

  Size of the margin (area between borders and screen edge) in cells.
- static final int MARGIN_INNER = CELL_SIZE ∗ MARGIN_CELLS

  Distance from screen edge to inner margin point.
- static final int MARGIN_OUTER = MARGIN_INNER - BORDER_THC

  Distance from screen edge to outer margin point (excluding the thickness).
- static final int MIN_CELL = MARGIN_CELLS

  Index of the first playable cell.
- static final int MAX_CELL = CELL_COUNT - MARGIN_CELLS - 1

  Index of the last playable cell.

6.9.1 Detailed Description

Defines constants used in the game.

Author

Maksims Orlovs

Definition at line 10 of file GameConstants.java.

6.9.2 Member Data Documentation

6.9.2.1 BORDER_THC

```
final int org.utilities.GameConstants.BORDER_THC = 5  [static]
```

Thickness of the borders in pixels.

Definition at line 41 of file GameConstants.java.

6.9.2.2 CELL_COUNT

```
final int org.utilities.GameConstants.CELL_COUNT = 40  [static]
```

The amount of cells in one row/column.

Definition at line 25 of file GameConstants.java.

6.9.2.3 CELL_SIZE

```
final int org.utilities.GameConstants.CELL_SIZE = WINDOW_SIZE.x / CELL_COUNT  [static]
```

Size of one cell.

Determined by the window size and the cell count.

Definition at line 30 of file GameConstants.java.

6.9.2.4 EFFECT_DURATION

```
final int org.utilities.GameConstants.EFFECT_DURATION = 8000  [static]
```

Duration of timed bonus food effects (in ms).

Definition at line 35 of file GameConstants.java.

6.9.2.5 FPS

```
final int org.utilities.GameConstants.FPS = 60  [static]
```

The base FPS of the game.

Definition at line 20 of file GameConstants.java.

6.9.2.6 MARGIN_CELLS

```
final int org.utilities.GameConstants.MARGIN_CELLS = 3  [static]
```

Size of the margin (area between borders and screen edge) in cells.

Unplayable area/cells.

Definition at line 46 of file GameConstants.java.

6.9.2.7 MARGIN_INNER

```
final int org.utilities.GameConstants.MARGIN_INNER = CELL_SIZE * MARGIN_CELLS  [static]
```

Distance from screen edge to inner margin point.

Determined by the size of the cells and size of the margin in cells.

Definition at line 52 of file GameConstants.java.

6.9.2.8 MARGIN_OUTER

```
final int org.utilities.GameConstants.MARGIN_OUTER = MARGIN_INNER - BORDER_THC  [static]
```

Distance from screen edge to outer margin point (excluding the thickness).

Definition at line 57 of file GameConstants.java.

6.9.2.9 MAX_CELL

```
final int org.utilities.GameConstants.MAX_CELL = CELL_COUNT - MARGIN_CELLS - 1  [static]
```

Index of the last playable cell.

Definition at line 67 of file GameConstants.java.

6.9.2.10 MIN_CELL

```
final int org.utilities.GameConstants.MIN_CELL = MARGIN_CELLS  [static]
```

Index of the first playable cell.

Definition at line 62 of file GameConstants.java.

6.9.2.11 WINDOW_SIZE

```
final Point org.utilities.GameConstants.WINDOW_SIZE = new Point(800, 800)  [static]
```

Point representing the window dimensions.

Definition at line 15 of file GameConstants.java.

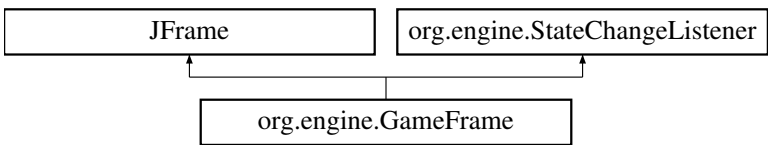
The documentation for this interface was generated from the following file:

- GameConstants.java

6.10 org.engine.GameFrame Class Reference

The GameFrame class is a part of the Game Engine system that handles switching and displaying appropriate game states.

Inheritance diagram for org.engine.GameFrame:



Public Member Functions

- GameFrame ()

  Creates a GameFrame and sets up the game, the window and the font.
- void changeState (GameState newState)

  Receives the call to switch to the next state, handles logic required for each state.

- void changeState (GameState newState)

Static Private Attributes

- static JPanel currentPanel

6.10.1 Detailed Description

The GameFrame class is a part of the Game Engine system that handles switching and displaying appropriate game states.

It extends JFrame to use Swing for displaying a game window and implements StateChangeListener to receive requests from the states to switch to a different state.

Author

Maksims Orlovs

Definition at line 19 of file GameFrame.java.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 GameFrame()

```
org.engine.GameFrame.GameFrame ( )
```

Creates a GameFrame and sets up the game, the window and the font.

Author

Maksims Orlovs
Halah Hasani (co-author)

Definition at line 27 of file GameFrame.java.

6.10.3 Member Function Documentation

6.10.3.1 changeState()

```
void org.engine.GameFrame.changeState (
            GameState newState )
```

Receives the call to switch to the next state, handles logic required for each state.

Parameters

newState	the next state of the game

Author

Maksims Orlovs

Implements org.engine.StateChangeListener.

Definition at line 58 of file GameFrame.java.

6.10.4 Member Data Documentation

6.10.4.1 currentPanel

```
JPanel org.engine.GameFrame.currentPanel  [static], [private]
```

Definition at line 20 of file GameFrame.java.

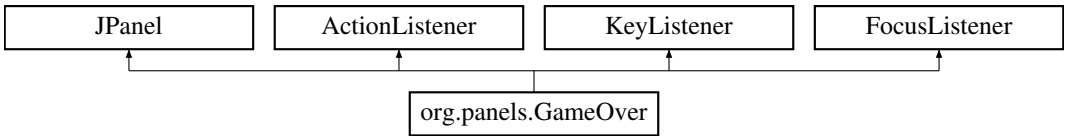
The documentation for this class was generated from the following file:

- GameFrame.java

6.11 org.panels.GameOver Class Reference

A panel that represents the game-over screen.

Inheritance diagram for org.panels.GameOver:



Public Member Functions

- GameOver (StateChangeListener listener, int score, boolean isHighScore)

  Constructs a game-over panel with the player's score.
- void actionPerformed (ActionEvent event)

  Defines the buttons' behaviour.
- void keyTyped (KeyEvent e)

  Limits text field to 3 characters and disallows typing special characters.
- void keyPressed (KeyEvent e)

  Request the observer to switch to the Leaderboard state when the name is entered and 'Enter' is pressed.
- void keyReleased (KeyEvent e)
- void focusGained (FocusEvent e)

  Remove the placeholder text from the text field when it gains focus.
- void focusLost (FocusEvent e)

Public Attributes

- BgPanel bg

Protected Member Functions

- void paintComponent (Graphics graphics)

  Draws the contents of the panel and the background.

Private Member Functions

- JTextField getjTextField ()

  Helper method to generate the text field.

Private Attributes

- final GameButton retryBtn
- final GameButton mainMenuBtn
- final ArrayList< GameButton > buttons
- StateChangeListener stateChanger
- JLabel scoreText
- JTextField enterNameField
- int score

6.11.1 Detailed Description

A panel that represents the game-over screen.

Author

Marwa Abohahcem

Definition at line 16 of file GameOver.java.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 GameOver()

```
org.panels.GameOver.GameOver (
            StateChangeListener listener,
            int score,
            boolean isHighScore )
```

Constructs a game-over panel with the player's score.

Changes depending on if the score is a new record. Displays a "Game Over" message, player's achieved score, buttons to retry and go to menu (if not a high score) or prompts the player to enter their name/initials (if is a high score).

Parameters

listener	reference to the observer class to allow state switching
score	player's score to be displayed
isHighScore	a flag to determine if the score is a new record

Author

Marwa Abohahcem

Victoria Rönnlid (co-author)

Definition at line 38 of file GameOver.java.

6.11.3 Member Function Documentation

6.11.3.1 actionPerformed()

```
void org.panels.GameOver.actionPerformed (
            ActionEvent event )
```

Defines the buttons' behaviour.

Parameters

event	the event to be processed

Author

Marwa Abohahcem

Definition at line 173 of file GameOver.java.

6.11.3.2 focusGained()

```
void org.panels.GameOver.focusGained (
            FocusEvent e )
```

Remove the placeholder text from the text field when it gains focus.

Parameters

e	the event to be processed

Author

Marwa Abohahcem

Definition at line 218 of file GameOver.java.

6.11.3.3 focusLost()

```
void org.panels.GameOver.focusLost (
            FocusEvent e )
```

Definition at line 225 of file GameOver.java.

6.11.3.4 getjTextField()

```
JTextField org.panels.GameOver.getjTextField ( )  [private]
```

Helper method to generate the text field.

Returns

JTextField object to be put onto the panel for the player's name prompt.

Author

Marwa Abohahcem

Definition at line 141 of file GameOver.java.

6.11.3.5 keyPressed()

```
void org.panels.GameOver.keyPressed (
            KeyEvent e )
```

Request the observer to switch to the Leaderboard state when the name is entered and 'Enter' is pressed.

Parameters

e	the key-press event to be processed

Author

Marwa Abohahcem

Definition at line 202 of file GameOver.java.

6.11.3.6 keyReleased()

```
void org.panels.GameOver.keyReleased (
            KeyEvent e )
```

Definition at line 210 of file GameOver.java.

6.11.3.7 keyTyped()

```
void org.panels.GameOver.keyTyped (
            KeyEvent e )
```

Limits text field to 3 characters and disallows typing special characters.

Parameters

e	the key-press event to be processed

Author

Marwa Abohahcem

Maksims Orlovs (co-author)

Definition at line 190 of file GameOver.java.

6.11.3.8 paintComponent()

```
void org.panels.GameOver.paintComponent (
            Graphics graphics )  [protected]
```

Draws the contents of the panel and the background.

Parameters

graphics	graphics component supplied by the GameFrame

Author

Marwa Abohahcem

Definition at line 162 of file GameOver.java.

6.11.4 Member Data Documentation

6.11.4.1 bg

```
BgPanel org.panels.GameOver.bg
```

Definition at line 19 of file GameOver.java.

6.11.4.2 buttons

```
final ArrayList<GameButton> org.panels.GameOver.buttons  [private]
```

Definition at line 20 of file GameOver.java.

6.11.4.3 enterNameField

```
JTextField org.panels.GameOver.enterNameField  [private]
```

Definition at line 25 of file GameOver.java.

6.11.4.4 mainMenuBtn

```
final GameButton org.panels.GameOver.mainMenuBtn  [private]
```

Definition at line 18 of file GameOver.java.

6.11.4.5 retryBtn

```
final GameButton org.panels.GameOver.retryBtn  [private]
```

Definition at line 17 of file GameOver.java.

6.11.4.6 score

```
int org.panels.GameOver.score  [private]
```

Definition at line 26 of file GameOver.java.

6.11.4.7 scoreText

```
JLabel org.panels.GameOver.scoreText  [private]
```

Definition at line 24 of file GameOver.java.

6.11.4.8 stateChanger

```
StateChangeListener org.panels.GameOver.stateChanger  [private]
```

Definition at line 22 of file GameOver.java.

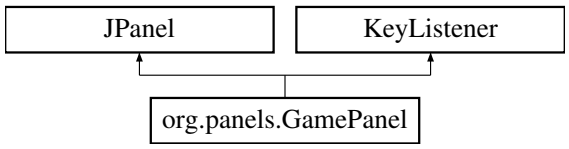
The documentation for this class was generated from the following file:

- GameOver.java

6.12 org.panels.GamePanel Class Reference

Represents the gameplay state.

Inheritance diagram for org.panels.GamePanel:



Public Member Functions

- GamePanel (StateChangeListener listener)
  Constructs the initial GamePanel and objects for handling the gameplay.
- void update ()
  Updates positions and interaction of all objects (snake effects, position, collision).
- int getScore ()
  Getter for the current player score.
- void paintComponent (Graphics g)
  Draws all contents of the panel (snake, food objects, obstacles, score) and the background.
- void startGame ()
  Starts the game loop.
- void stopGame ()
  Ends the game loop and changes to the appropriate game-over screen.
- void keyTyped (KeyEvent e)
- void keyPressed (KeyEvent e)
  Handles user input.
- void keyReleased (KeyEvent e)

Private Member Functions

- void adjustSnakeSpeed (double speedMultiplier)

  Adjusts the delay between frames in the game loop.
- void updateEffects ()

  Updates the status of the effects.
- boolean doFoodCollisions ()

  Checks collisions with food items.
- void applyFoodEffect (FoodType foodType)

  Applies the effect of an eaten food object to the game depending on the type of the food.
- Food generateNewFoodItem (boolean isBonus)

  Generates a food object in a random valid position.

Private Attributes

- BgPanel bg
- Snake snake
- ArrayList< Food > food
- Random rand
- ObstacleList obstacles
- int score
- final Timer gameLoop
- long startTime
- boolean fastMode
- boolean slowMode
- boolean keyInverter
- StateChangeListener stateChanger

6.12.1 Detailed Description

Represents the gameplay state.

Handles gameplay logic and drawing of all objects.

Author

Maksims Orlovs

Fatemeh Akbarifar (co-author)

Definition at line 27 of file GamePanel.java.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 GamePanel()

```
org.panels.GamePanel.GamePanel (
            StateChangeListener listener )
```

Constructs the initial GamePanel and objects for handling the gameplay.

Parameters

listener	reference to the observer object to allow requesting state change

Author

Maksims Orlovs

Fatemeh Akbarifar (co-author)

Definition at line 47 of file GamePanel.java.

6.12.3 Member Function Documentation

6.12.3.1 adjustSnakeSpeed()

```
void org.panels.GamePanel.adjustSnakeSpeed (
            double speedMultiplier )  [private]
```

Adjusts the delay between frames in the game loop.

Appears as the game's speed change.

Parameters

speedMultiplier	factor to multiply the speed by.

Author

Fatemeh Akbarifar

Definition at line 111 of file GamePanel.java.

6.12.3.2 applyFoodEffect()

```
void org.panels.GamePanel.applyFoodEffect (
            FoodType foodType )  [private]
```

Applies the effect of an eaten food object to the game depending on the type of the food.

Parameters

foodType	the type of the eaten food

Author

Fatemeh Akbarifar

Maksims Orlovs (co-author)

Marwa Abohachem (co-author)

Definition at line 176 of file GamePanel.java.

6.12.3.3 doFoodCollisions()

```
boolean org.panels.GamePanel.doFoodCollisions ( )  [private]
```

Checks collisions with food items.

Returns

true if food is eaten

Author

Fatemeh Akbarifar

Maksims Orlovs (co-author)

Definition at line 139 of file GamePanel.java.

6.12.3.4 generateNewFoodItem()

```
Food org.panels.GamePanel.generateNewFoodItem (
            boolean isBonus )  [private]
```

Generates a food object in a random valid position.

Parameters

isBonus	determines if a food item to generate is default or bonus

Returns

generated Food object

Author

Maksims Orlovs

Definition at line 215 of file GamePanel.java.

6.12.3.5 getScore()

```
int org.panels.GamePanel.getScore ( )
```

Getter for the current player score.

Returns

current score

Author

Fatemeh Akbarifar

Definition at line 234 of file GamePanel.java.

6.12.3.6 keyPressed()

```
void org.panels.GamePanel.keyPressed (
            KeyEvent e )
```

Handles user input.

Requests the change of direction from the snake if one of the arrow keys is pressed.

Parameters

e	the key-press event to be processed

Author

Fatemeh Akbarifar

Maksims Orlovs (co-author)

Definition at line 310 of file GamePanel.java.

6.12.3.7 keyReleased()

```
void org.panels.GamePanel.keyReleased (
            KeyEvent e )
```

Definition at line 348 of file GamePanel.java.

6.12.3.8 keyTyped()

```
void org.panels.GamePanel.keyTyped (
            KeyEvent e )
```

Definition at line 301 of file GamePanel.java.

6.12.3.9 paintComponent()

```
void org.panels.GamePanel.paintComponent (
            Graphics g )
```

Draws all contents of the panel (snake, food objects, obstacles, score) and the background.

Parameters

g	graphics component supplied by the GameFrame

Author

Maksims Orlovs
Fatemeh Akbarifar (co-author)
Marwa Abohachem (co-author)

Definition at line 246 of file GamePanel.java.

6.12.3.10 startGame()

```
void org.panels.GamePanel.startGame ( )
```

Starts the game loop.

Author

Maksims Orlovs

Definition at line 279 of file GamePanel.java.

6.12.3.11 stopGame()

```
void org.panels.GamePanel.stopGame ( )
```

Ends the game loop and changes to the appropriate game-over screen.

Author

Fatemeh Akbarifar

Definition at line 287 of file GamePanel.java.

6.12.3.12 update()

```
void org.panels.GamePanel.update ( )
```

Updates positions and interaction of all objects (snake effects, position, collision).

Part of game loop.

Author

Maksims Orlovs
Fatemeh Akbarifar (co-author)

Definition at line 82 of file GamePanel.java.

6.12.3.13 updateEffects()

```
void org.panels.GamePanel.updateEffects ( )  [private]
```

Updates the status of the effects.

Check if the effect time is over, removes the effect.

Author

Fatemeh Akbarifar

Definition at line 120 of file GamePanel.java.

6.12.4 Member Data Documentation

6.12.4.1 bg

```
BgPanel org.panels.GamePanel.bg  [private]
```

Definition at line 28 of file GamePanel.java.

6.12.4.2 fastMode

```
boolean org.panels.GamePanel.fastMode  [private]
```

Definition at line 38 of file GamePanel.java.

6.12.4.3 food

```
ArrayList<Food> org.panels.GamePanel.food  [private]
```

Definition at line 30 of file GamePanel.java.

6.12.4.4 gameLoop

```
final Timer org.panels.GamePanel.gameLoop  [private]
```

Definition at line 35 of file GamePanel.java.

6.12.4.5 keyInverter

```
boolean org.panels.GamePanel.keyInverter  [private]
```

Definition at line 38 of file GamePanel.java.

6.12.4.6 obstacles

```
ObstacleList org.panels.GamePanel.obstacles  [private]
```

Definition at line 32 of file GamePanel.java.

6.12.4.7 rand

```
Random org.panels.GamePanel.rand  [private]
```

Definition at line 31 of file GamePanel.java.

6.12.4.8 score

```
int org.panels.GamePanel.score  [private]
```

Definition at line 34 of file GamePanel.java.

6.12.4.9 slowMode

```
boolean org.panels.GamePanel.slowMode  [private]
```

Definition at line 38 of file GamePanel.java.

6.12.4.10 snake

```
Snake org.panels.GamePanel.snake  [private]
```

Definition at line 29 of file GamePanel.java.

6.12.4.11 startTime

```
long org.panels.GamePanel.startTime  [private]
```

Definition at line 36 of file GamePanel.java.

6.12.4.12 stateChanger

```
StateChangeListener org.panels.GamePanel.stateChanger  [private]
```

Definition at line 39 of file GamePanel.java.

The documentation for this class was generated from the following file:

- GamePanel.java

6.13 org.engine.GameState Enum Reference

Represents all possible game states.

Public Attributes

- MENU
- GAME
- GAME_OVER
- GAME_OVER_ENTERNAME
- LEADERBOARD
- TUTORIAL

6.13.1 Detailed Description

Represents all possible game states.

Author

Maksims Orlovs

Definition at line 6 of file GameState.java.

6.13.2 Member Data Documentation

6.13.2.1 GAME

`org.engine.GameState.GAME`

Definition at line 7 of file GameState.java.

6.13.2.2 GAME_OVER

`org.engine.GameState.GAME_OVER`

Definition at line 7 of file GameState.java.

6.13.2.3 GAME_OVER_ENTERNAME

`org.engine.GameState.GAME_OVER_ENTERNAME`

Definition at line 7 of file GameState.java.

6.13.2.4 LEADERBOARD

`org.engine.GameState.LEADERBOARD`

Definition at line 7 of file GameState.java.

6.13.2.5 MENU

`org.engine.GameState.MENU`

Definition at line 7 of file GameState.java.

6.13.2.6 TUTORIAL

`org.engine.GameState.TUTORIAL`

Definition at line 8 of file GameState.java.

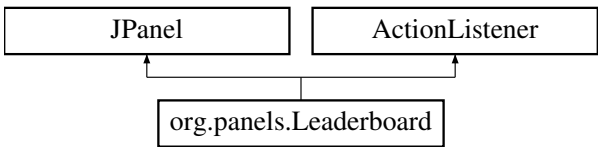
The documentation for this enum was generated from the following file:

- GameState.java

6.14 org.panels.Leaderboard Class Reference

Panel representing the Leaderboard screen.

Inheritance diagram for org.panels.Leaderboard:



Public Member Functions

- Leaderboard (StateChangeListener listener)

  Creates the Leaderboard panel.
- void paintComponent (Graphics g)

  Draws all the contents of the panel and the background.
- void actionPerformed (ActionEvent event)

  Defines button behaviour.

Static Public Member Functions

- static void createPlayer (String name, long score)

  Creates and appends a new player to the leaderboard file.
- static boolean isTopTen (Player playerInTop10)

  Checks if players score is among top 10.

Private Member Functions

- void readToList ()

  Adds first 10 top scoring players from the file to the list for display.

Static Private Member Functions

- static ArrayList< Player > readFromFile ()

  Returns a sorted list of all players read from the json file.

Private Attributes

- DefaultListModel< String > listItems
- JList< String > lbList
- GameButton mainMenuBtn
- BgPanel bg
- StateChangeListener stateChanger

Static Private Attributes

- static ArrayList< Player > playerList = new ArrayList<>()

6.14.1 Detailed Description

Panel representing the Leaderboard screen.

Author

Halah Hasani

Definition at line 28 of file Leaderboard.java.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Leaderboard()

```
org.panels.Leaderboard.Leaderboard (
            StateChangeListener listener )
```

Creates the Leaderboard panel.

Reads the stored leaderboard data from file.

Parameters

listener	reference to the observer object to allow requesting state change

Author

Halah Hasani

Victoria Rönnlid (co-author)

Marwa Abohahcem (co-author)

Definition at line 45 of file Leaderboard.java.

6.14.3 Member Function Documentation

6.14.3.1 actionPerformed()

```
void org.panels.Leaderboard.actionPerformed (
            ActionEvent event )
```

Defines button behaviour.

Parameters

event	the button-press event to be processed

Author

Halah Hasani

Definition at line 97 of file Leaderboard.java.

6.14.3.2 createPlayer()

```
static void org.panels.Leaderboard.createPlayer (
            String name,
            long score )  [static]
```

Creates and appends a new player to the leaderboard file.

Completely rewrites the json file.

Parameters

name	player's name
score	player's score

Author

Halah Hasani

Definition at line 170 of file Leaderboard.java.

6.14.3.3 isTopTen()

```
static boolean org.panels.Leaderboard.isTopTen (
            Player playerInTop10 )  [static]
```

Checks if players score is among top 10.

Parameters

playerInTop10	player object to be checked against the leaderboard file

Returns

true if the player qualifies for the leaderboard

Author

Halah Hasani

Maksims Orlovs (co-author)

Definition at line 204 of file Leaderboard.java.

6.14.3.4 paintComponent()

```
void org.panels.Leaderboard.paintComponent (
            Graphics g )
```

Draws all the contents of the panel and the background.

Parameters

g	graphics component supplied by the GameFrame

Definition at line 86 of file Leaderboard.java.

6.14.3.5 readFromFile()

```
static ArrayList< Player > org.panels.Leaderboard.readFromFile ( )  [static], [private]
```

Returns a sorted list of all players read from the json file.

Creates an empty JSON file if it is not present.

Returns

sorted ArrayList of all Player:s stored on disk

Author

Halah Hasani

Definition at line 110 of file Leaderboard.java.

6.14.3.6 readToList()

```
void org.panels.Leaderboard.readToList ( )  [private]
```

Adds first 10 top scoring players from the file to the list for display.

Formats the list elements according to the design.

Author

Halah Hasani

Definition at line 148 of file Leaderboard.java.

6.14.4 Member Data Documentation

6.14.4.1 bg

```
BgPanel org.panels.Leaderboard.bg  [private]
```

Definition at line 32 of file Leaderboard.java.

6.14.4.2 lbList

```
JList<String> org.panels.Leaderboard.lbList  [private]
```

Definition at line 30 of file Leaderboard.java.

6.14.4.3 listItems

```
DefaultListModel<String> org.panels.Leaderboard.listItems  [private]
```

Definition at line 29 of file Leaderboard.java.

6.14.4.4 mainMenuBtn

```
GameButton org.panels.Leaderboard.mainMenuBtn  [private]
```

Definition at line 31 of file Leaderboard.java.

6.14.4.5 playerList

```
ArrayList<Player> org.panels.Leaderboard.playerList = new ArrayList<>()  [static], [private]
```

Definition at line 34 of file Leaderboard.java.

6.14.4.6 stateChanger

```
StateChangeListener org.panels.Leaderboard.stateChanger  [private]
```

Definition at line 36 of file Leaderboard.java.

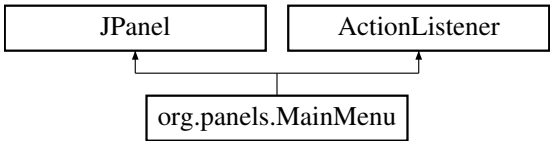
The documentation for this class was generated from the following file:

- Leaderboard.java

6.15 org.panels.MainMenu Class Reference

A panel representing the main menu.

Inheritance diagram for org.panels.MainMenu:



Public Member Functions

- MainMenu (StateChangeListener listener)
  Creates the menu object.
- void paintComponent (Graphics g)
  Draws all menu elements and the background.
- void actionPerformed (ActionEvent event)
  Defines button behaviour.

Public Attributes

- BgPanel bg

Private Attributes

- GameButton startBtn
- GameButton tutorialBtn
- GameButton leaderboardBtn
- GameButton exitBtn
- ArrayList< GameButton > buttons
- StateChangeListener stateChanger

6.15.1 Detailed Description

A panel representing the main menu.

Author

Victoria Rönnlid

Definition at line 18 of file MainMenu.java.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 MainMenu()

```
org.panels.MainMenu.MainMenu (
            StateChangeListener listener )
```

Creates the menu object.

Parameters

listener	reference to the observer object to allow requesting state change

Author

Victoria Rönnlid

Definition at line 35 of file MainMenu.java.

6.15.3 Member Function Documentation

6.15.3.1 actionPerformed()

```
void org.panels.MainMenu.actionPerformed (
            ActionEvent event )
```

Defines button behaviour.

Parameters

event	the button-press event to be processed

Author

Victoria Rönnlid

Definition at line 86 of file MainMenu.java.

6.15.3.2 paintComponent()

```
void org.panels.MainMenu.paintComponent (
            Graphics g )
```

Draws all menu elements and the background.

Parameters

g	graphics component supplied by the GameFrame

Author

Victoria Rönnlid

Definition at line 75 of file MainMenu.java.

6.15.4 Member Data Documentation

6.15.4.1 bg

```
BgPanel org.panels.MainMenu.bg
```

Definition at line 24 of file MainMenu.java.

6.15.4.2 buttons

```
ArrayList<GameButton> org.panels.MainMenu.buttons  [private]
```

Definition at line 26 of file MainMenu.java.

6.15.4.3 exitBtn

```
GameButton org.panels.MainMenu.exitBtn  [private]
```

Definition at line 23 of file MainMenu.java.

6.15.4.4 leaderboardBtn

```
GameButton org.panels.MainMenu.leaderboardBtn  [private]
```

Definition at line 22 of file MainMenu.java.

6.15.4.5 startBtn

```
GameButton org.panels.MainMenu.startBtn  [private]
```

Definition at line 19 of file MainMenu.java.

6.15.4.6 stateChanger

```
StateChangeListener org.panels.MainMenu.stateChanger  [private]
```

Definition at line 28 of file MainMenu.java.

6.15.4.7 tutorialBtn

```
GameButton org.panels.MainMenu.tutorialBtn  [private]
```

Definition at line 21 of file MainMenu.java.

The documentation for this class was generated from the following file:

- MainMenu.java

6.16 org.objects.obstacle.Obstacle Class Reference

Represents an obstacle.

Public Member Functions

- Obstacle (ArrayList< CellPosition > snakePos)
  Creates and spawnds an obstacle object.
- void respawn (ArrayList< CellPosition > snakePos)
  Spawns the obstacle of a random size and shape on the playing field, in a valid position.
- void draw (Graphics2D frame)
  Method to draw the obstacle item onto the screen (frame).
- ArrayList< CellPosition > getCells ()
  Getter for the total obstacle position.

Private Member Functions

- CellPosition getRandomCell ()
  Generates a random cell within the playable area (excludes margins)

Private Attributes

- ArrayList< CellPosition > cells
- Random rand

Static Private Attributes

- static final int MAX_SIZE = 5
- static final int PARTICLE_COUNT = 8
- static final int PARTICLE_SIZE = GameConstants.CELL_SIZE / 5

6.16.1 Detailed Description

Represents an obstacle.

Author

Maksims Orlovs

Definition at line 14 of file Obstacle.java.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Obstacle()

```
org.objects.obstacle.Obstacle.Obstacle (
            ArrayList< CellPosition > snakePos )
```

Creates and spawnds an obstacle object.

Author

Maksims Orlovs

Parameters

snakePos	snake position at the time of obstacle spawning to prevent incorrect spawning position.

See also

Obstacle::respawn(ArrayList)

Definition at line 28 of file Obstacle.java.

6.16.3 Member Function Documentation

6.16.3.1 draw()

```
void org.objects.obstacle.Obstacle.draw (
            Graphics2D frame )
```

Method to draw the obstacle item onto the screen (frame).

Draws squares in obstacle positions and random "noise" inside of them.

Parameters

frame	Swing Graphics2D object that represents the current frame to be updated.

Author

Maksims Orlovs

Definition at line 97 of file Obstacle.java.

6.16.3.2 getCells()

```
ArrayList< CellPosition > org.objects.obstacle.Obstacle.getCells ( )
```

Getter for the total obstacle position.

Returns

ArrayList of CellPosition:s that represent the cells occupied by the obstacle.

Author

Maksims Orlovs

Definition at line 124 of file Obstacle.java.

6.16.3.3 getRandomCell()

```
CellPosition org.objects.obstacle.Obstacle.getRandomCell ( )  [private]
```

Generates a random cell within the playable area (excludes margins)

Returns

CellPosition representing a traversable cell

Author

Fatemeh Akbarifar

Definition at line 85 of file Obstacle.java.

6.16.3.4 respawn()

```
void org.objects.obstacle.Obstacle.respawn (
            ArrayList< CellPosition > snakePos )
```

Spawns the obstacle of a random size and shape on the playing field, in a valid position.

Spawning stops when an incorrect position has been generated.

Parameters

snakePos	snake position at the time of obstacle spawning to prevent spawning in the snake.

Author

Maksims Orlovs

Definition at line 40 of file Obstacle.java.

6.16.4 Member Data Documentation

6.16.4.1 cells

```
ArrayList<CellPosition> org.objects.obstacle.Obstacle.cells  [private]
```

Definition at line 19 of file Obstacle.java.

6.16.4.2 MAX_SIZE

```
final int org.objects.obstacle.Obstacle.MAX_SIZE = 5  [static], [private]
```

Definition at line 15 of file Obstacle.java.

6.16.4.3 PARTICLE_COUNT

```
final int org.objects.obstacle.Obstacle.PARTICLE_COUNT = 8  [static], [private]
```

Definition at line 16 of file Obstacle.java.

6.16.4.4 PARTICLE_SIZE

```
final int org.objects.obstacle.Obstacle.PARTICLE_SIZE = GameConstants.CELL_SIZE / 5  [static],
[private]
```

Definition at line 17 of file Obstacle.java.

6.16.4.5 rand

```
Random org.objects.obstacle.Obstacle.rand  [private]
```

Definition at line 20 of file Obstacle.java.

The documentation for this class was generated from the following file:

- Obstacle.java

6.17 org.objects.obstacle.ObstacleList Class Reference

Represents a List of all existing obstacles in the game.

Public Member Functions

- ObstacleList ()
  Constructs the ObstacleList with an empty ArrayList of Obstacle.
- ArrayList< CellPosition > getAllCells ()
  Returns a sum of positions of all obstacles.
- void add (Obstacle obstacle)
  A setter to add an Obstacle to the list.
- ArrayList< Obstacle > getObstacles ()
  A getter for the list of obstacles.

Private Attributes

- ArrayList< Obstacle > obstacles

6.17.1 Detailed Description

Represents a List of all existing obstacles in the game.

Author

Maksims Orlovs

Definition at line 11 of file ObstacleList.java.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 ObstacleList()

```
org.objects.obstacle.ObstacleList.ObstacleList ( )
```

Constructs the ObstacleList with an empty ArrayList of Obstacle.

Author

Maksims Orlovs

Definition at line 18 of file ObstacleList.java.

6.17.3 Member Function Documentation

6.17.3.1 add()

```
void org.objects.obstacle.ObstacleList.add (
            Obstacle obstacle )
```

A setter to add an Obstacle to the list.

Parameters

obstacle	new Obstacle to be added to the list

Author

Maksims Orlovs

Definition at line 40 of file ObstacleList.java.

6.17.3.2 getAllCells()

```
ArrayList< CellPosition > org.objects.obstacle.ObstacleList.getAllCells ( )
```

Returns a sum of positions of all obstacles.

Returns

An ArrayList of CellPosition:s that represents a list of all cells occupied by all obstacles in the list.

Author

Maksims Orlovs

Definition at line 27 of file ObstacleList.java.

6.17.3.3 getObstacles()

```
ArrayList< Obstacle > org.objects.obstacle.ObstacleList.getObstacles ( )
```

A getter for the list of obstacles.

Returns

returns a list of all currently present Obstacles

Author

Maksims Orlovs

Definition at line 49 of file ObstacleList.java.

6.17.4 Member Data Documentation

6.17.4.1 obstacles

```
ArrayList<Obstacle> org.objects.obstacle.ObstacleList.obstacles  [private]
```

Definition at line 12 of file ObstacleList.java.

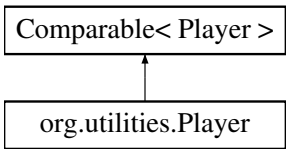
The documentation for this class was generated from the following file:

- ObstacleList.java

6.18 org.utilities.Player Class Reference

Represents a player that can be added to the leaderboard.

Inheritance diagram for org.utilities.Player:



Public Member Functions

- Player (String name, long score)
  Creates a player object with specified parameters.
- String getName ()
  Getter for the Player's name.
- long getScore ()
  Getter for the Player's score.
- String getNamesAndScores ()
  Creates a String representing a line in the leaderboard.
- boolean equals (Object o)
  Compares the Player to another object.
- int compareTo (Player other)
  Compares the Player's score to another Player's score.

Private Attributes

- String name
- long score

6.18.1 Detailed Description

Represents a player that can be added to the leaderboard.

Author

Halah Hasani

Definition at line 8 of file Player.java.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Player()

```
org.utilities.Player.Player (
            String name,
            long score )
```

Creates a player object with specified parameters.

Parameters

name	player's name
score	player's score

Author

Halah Hasani

Definition at line 19 of file Player.java.

6.18.3 Member Function Documentation

6.18.3.1 compareTo()

```
int org.utilities.Player.compareTo (
            Player other )
```

Compares the Player's score to another Player's score.

Parameters

other	the player to be compared.

Returns

1 if other's score is higher, -1 if other's score is lower. Returns result of two players' name lexicographic comparison if scores are equal.

Author

Maskims Orlovs

Definition at line 80 of file Player.java.

6.18.3.2 equals()

```
boolean org.utilities.Player.equals (
            Object o )
```

Compares the Player to another object.

Parameters

o	an object to be compared with this object

Returns

true if names and scores of both players are equal

Author

Maskims Orlovs

Definition at line 62 of file Player.java.

6.18.3.3 getName()

```
String org.utilities.Player.getName ( )
```

Getter for the Player's name.

Returns

player's name

Author

Halah Hasani

Definition at line 29 of file Player.java.

6.18.3.4 getNamesAndScores()

```
String org.utilities.Player.getNamesAndScores ( )
```

Creates a String representing a line in the leaderboard.

Formatted according to the design (ABC---—1).

Returns

Leaderboard line with the player's name and score

Author

Halah Hasani

Definition at line 47 of file Player.java.

6.18.3.5 getScore()

```
long org.utilities.Player.getScore ( )
```

Getter for the Player's score.

Returns

player's score

Author

Halah Hasani

Definition at line 38 of file Player.java.

6.18.4 Member Data Documentation

6.18.4.1 name

```
String org.utilities.Player.name  [private]
```

Definition at line 10 of file Player.java.

6.18.4.2 score

```
long org.utilities.Player.score  [private]
```

Definition at line 11 of file Player.java.

The documentation for this class was generated from the following file:

- Player.java

6.19 org.objects.Snake Class Reference

Represents a snake object.

Public Member Functions

- Snake ()

  Constructs a Snake object in the middle of the screen.
- ArrayList< CellPosition > getBody ()

  Getter for the current snake position.
- boolean doCollisions ()

  Checks snake's collision with self and borders.
- void move ()

  Updates the position of the head depending on the direction input.
- void updateDirection (Direction newDir)

  Adds new input to the input queue if less than 2 inputs are queued and if is not opposite to the previously queued input.
- void draw (Graphics2D frame)

  Method to draw the snake object onto the screen (frame).
- boolean checkCollisionWith (CellPosition pos)

  Checks if snake collides with any object at given position.
- boolean checkCollisionWith (ArrayList< CellPosition > pos)

  Checks if snake collides with a multi-cell object at given position.

Static Public Attributes

- static final double SPEED = 0.18
- static final int INIT_LEN = 5

Private Member Functions

- CellPosition calculateNextPos ()

  Helper method for calculating the position of the head in after applying the movement.
- boolean doSelfCollision (CellPosition head)

  Helper method that checks if the snake collided with itself.
- boolean doBorderCollision (CellPosition head)

  Helper method that checks if the snake collided with itself.
- boolean isOppositeDir (Direction dir1, Direction dir2)

  Checks if two supplied directions are opposite to each other.

Private Attributes

- ArrayList< CellPosition > body
- Direction currentDirection
- LinkedList< Direction > inputQueue

6.19.1 Detailed Description

Represents a snake object.

Author

Maksims Orlovs

Definition at line 15 of file Snake.java.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Snake()

```
org.objects.Snake.Snake ( )
```

Constructs a Snake object in the middle of the screen.

Initial parameters: length 5, direction right.

Author

Maksims Orlovs

Definition at line 27 of file Snake.java.

6.19.3 Member Function Documentation

6.19.3.1 calculateNextPos()

```
CellPosition org.objects.Snake.calculateNextPos ( )  [private]
```

Helper method for calculating the position of the head in after applying the movement.

Returns

a CellPosition representing the cell that the Snake's head is to be moved to.

Author

Maksims Orlovs

Definition at line 52 of file Snake.java.

6.19.3.2 checkCollisionWith() [1/2]

```
boolean org.objects.Snake.checkCollisionWith (
            ArrayList< CellPosition > pos )
```

Checks if snake collides with a multi-cell object at given position.

Overload to support multi-cell objects.

Parameters

pos	an ArrayList of CellPosition:s that represents a multi-cell object to check collision with.

Returns

true if the snake collides with any cell of the multi-cell object.

Author

Maksims Orlovs

Definition at line 176 of file Snake.java.

6.19.3.3 checkCollisionWith() [2/2]

```
boolean org.objects.Snake.checkCollisionWith (
            CellPosition pos )
```

Checks if snake collides with any object at given position.

Parameters

pos	a CellPosition of an object to check collision with.

Returns

true if the snake collides with the object.

Author

Maksims Orlovs

Definition at line 166 of file Snake.java.

6.19.3.4 doBorderCollision()

```
boolean org.objects.Snake.doBorderCollision (
            CellPosition head )  [private]
```

Helper method that checks if the snake collided with itself.

Parameters

head	the CellPosition representing the Snake's head position

Returns

true if snake's head collided with one of the borders

Author

Fatemeh Akbarifar

Definition at line 83 of file Snake.java.

6.19.3.5 doCollisions()

```
boolean org.objects.Snake.doCollisions ( )
```

Checks snake's collision with self and borders.

Returns

true if the snake collided with self or one of the borders.

See also

Snake::doSelfCollision(CellPosition)

Snake::doBorderCollision(CellPosition)

Author

Maksims Orlovs

Definition at line 101 of file Snake.java.

6.19.3.6 doSelfCollision()

```
boolean org.objects.Snake.doSelfCollision (
            CellPosition head )  [private]
```

Helper method that checks if the snake collided with itself.

Parameters

head	the CellPosition representing the Snake's head position

Returns

true if snake's head collided with its body

Author

Maksims Orlovs

Definition at line 72 of file Snake.java.

6.19.3.7 draw()

```
void org.objects.Snake.draw (
            Graphics2D frame )
```

Method to draw the snake object onto the screen (frame).

Parameters

frame	Swing Graphics2D object that represents the current frame to be updated.

Author

Maksims Orlovs

Definition at line 152 of file Snake.java.

6.19.3.8 getBody()

```
ArrayList< CellPosition > org.objects.Snake.getBody ( )
```

Getter for the current snake position.

Returns

an ArrayList of CellPosition:s representing all cells occupied by the Snake body.

Author

Maksims Orlovs

Definition at line 43 of file Snake.java.

6.19.3.9 isOppositeDir()

```
boolean org.objects.Snake.isOppositeDir (
            Direction dir1,
            Direction dir2 )  [private]
```

Checks if two supplied directions are opposite to each other.

Parameters

dir1	Direction 1
dir2	Direction 2

Returns

true if the directions are opposite

Author

Maksims Orlovs

Definition at line 140 of file Snake.java.

6.19.3.10 move()

```
void org.objects.Snake.move ( )
```

Updates the position of the head depending on the direction input.

Adds the new head to the snake body.

See also

Snake::calculateNextPos()

Author

Maksims Orlovs

Definition at line 111 of file Snake.java.

6.19.3.11 updateDirection()

```
void org.objects.Snake.updateDirection (
            Direction newDir )
```

Adds new input to the input queue if less than 2 inputs are queued and if is not opposite to the previously queued input.

Parameters

newDir	new direction requested from the user (from the keyboard input in the engine)

Author

Maksims Orlovs

Definition at line 125 of file Snake.java.

6.19.4 Member Data Documentation

6.19.4.1 body

```
ArrayList<CellPosition> org.objects.Snake.body  [private]
```

Definition at line 19 of file Snake.java.

6.19.4.2 currentDirection

```
Direction org.objects.Snake.currentDirection  [private]
```

Definition at line 20 of file Snake.java.

6.19.4.3 INIT_LEN

```
final int org.objects.Snake.INIT_LEN = 5  [static]
```

Definition at line 17 of file Snake.java.

6.19.4.4 inputQueue

```
LinkedList<Direction> org.objects.Snake.inputQueue  [private]
```

Definition at line 21 of file Snake.java.

6.19.4.5 SPEED

```
final double org.objects.Snake.SPEED = 0.18  [static]
```

Definition at line 16 of file Snake.java.

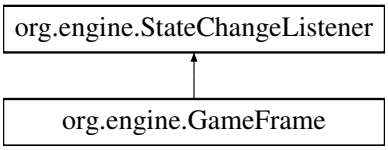
The documentation for this class was generated from the following file:

- Snake.java

6.20 org.engine.StateChangeListener Interface Reference

Interface to allow state switching in the engine from the states (Observer).

Inheritance diagram for org.engine.StateChangeListener:



Public Member Functions

- void changeState (GameState newState)

6.20.1 Detailed Description

Interface to allow state switching in the engine from the states (Observer).

Author

Maksims Orlovs

Definition at line 7 of file StateChangeListener.java.

6.20.2 Member Function Documentation

6.20.2.1 changeState()

```
void org.engine.StateChangeListener.changeState (
            GameState newState )
```

Implemented in org.engine.GameFrame.

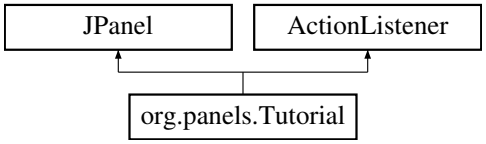
The documentation for this interface was generated from the following file:

- StateChangeListener.java

6.21 org.panels.Tutorial Class Reference

Represents a tutorial screen.

Inheritance diagram for org.panels.Tutorial:



Public Member Functions

- Tutorial (StateChangeListener listener)

  Creates the screen, loads tutorial image and applies layout.
- void paintComponent (Graphics g)
- void actionPerformed (ActionEvent event)

Private Attributes

- GameButton menuBtn
- BufferedImage tutorialPic
- StateChangeListener stateChanger

6.21.1 Detailed Description

Represents a tutorial screen.

Author

Victoria Rönnlid

Definition at line 21 of file Tutorial.java.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Tutorial()

```
org.panels.Tutorial.Tutorial (
            StateChangeListener listener )
```

Creates the screen, loads tutorial image and applies layout.

Author

Victoria Rönnlid

Maksims Orlovs (co-author)

Parameters

listener	

Definition at line 33 of file Tutorial.java.

6.21.3 Member Function Documentation

6.21.3.1 actionPerformed()

```
void org.panels.Tutorial.actionPerformed (
            ActionEvent event )
```

Definition at line 59 of file Tutorial.java.

6.21.3.2 paintComponent()

```
void org.panels.Tutorial.paintComponent (
            Graphics g )
```

Definition at line 53 of file Tutorial.java.

6.21.4 Member Data Documentation

6.21.4.1 menuBtn

```
GameButton org.panels.Tutorial.menuBtn  [private]
```

Definition at line 22 of file Tutorial.java.

6.21.4.2 stateChanger

```
StateChangeListener org.panels.Tutorial.stateChanger  [private]
```

Definition at line 25 of file Tutorial.java.

6.21.4.3 tutorialPic

```
BufferedImage org.panels.Tutorial.tutorialPic  [private]
```

Definition at line 24 of file Tutorial.java.

The documentation for this class was generated from the following file:

- Tutorial.java

Chapter 7

File Documentation

7.1 App.java File Reference

Classes

- class org.app.App

  Class to launch the game.

Packages

- package org.app

7.2 App.java

Go to the documentation of this file.

```
00001 package org.app;
00002
00006 public class App {
00007     public static void main(String[] args) {
00008         new org.engine.GameFrame();
00009     }
00010 }
```

7.3 GameFrame.java File Reference

Classes

- class org.engine.GameFrame

  The GameFrame class is a part of the Game Engine system that handles switching and displaying appropriate game states.

Packages

- package org.engine

7.4 GameFrame.java

Go to the documentation of this file.

```
00001 package org.engine;
00002
00003 import javax.swing.*;
00004 import java.awt.*;
00005 import java.io.File;
00006 import java.io.IOException;
00007
00008 import org.utilities.GameConstants;
00009 import org.panels.*;
00010
00011
00019 public class GameFrame extends JFrame implements StateChangeListener {
00020     private static JPanel currentPanel;
00021
00027     public GameFrame() {
00028         super();
00029
00030         // window setup
00031         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
00032         this.setResizable(false);
00033         this.setTitle("Snake Evolution");
00034         this.setSize(GameConstants.WINDOW_SIZE.x, GameConstants.WINDOW_SIZE.y);
00035
00036         this.setLocationRelativeTo(null);
00037         this.setFocusable(true);
00038
00039         // create font for use in all panels
00040         try {
00041             Font gameFont = Font.createFont(Font.TRUETYPE_FONT, new File("assets/PublicPixel.ttf"));
00042             GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
00043             ge.registerFont(gameFont); // makes the font available to font constructors by font name
00044         } catch (FontFormatException | IOException e) {
00045             throw new RuntimeException("Font creation error\n" + e);
00046         }
00047
00048         changeState(GameState.MENU);
00049         this.setVisible(true);
00050     }
00051
00057     @Override
00058     public void changeState(GameState newState) {
00059         getContentPane().removeAll();
00060         switch (newState) {
00061             case MENU -> {
00062                 currentPanel = new MainMenu(this);
00063             }
00064
00065             case GAME -> {
00066                 GamePanel gamePanel = new GamePanel(this);
00067                 this.addKeyListener(gamePanel);
00068                 gamePanel.requestFocusInWindow();
00069                 currentPanel = gamePanel;
00070                 gamePanel.startGame();
00071             }
00072
00073             case GAME_OVER -> {
00074                 int score = ((GamePanel)currentPanel).getScore(); // casting is safe, previous panel
      guaranteed to be panels.GamePanel
00075                 GameOver nextPanel = new GameOver(this, score, false);
00076                 currentPanel = nextPanel;
00077             }
00078
00079             case GAME_OVER_ENTERNAME -> {
00080                 int score = ((GamePanel)currentPanel).getScore();
00081                 GameOver nextPanel = new GameOver(this, score, true);
00082                 currentPanel = nextPanel;
00083             }
00084
00085             case LEADERBOARD -> {
00086                 currentPanel = new Leaderboard(this);
00087             }
00088             case TUTORIAL -> {
00089                 currentPanel = new Tutorial(this);
00090             }
00091         }
00092         this.add(currentPanel);
00093         this.pack();
00094     }
00095 }
```

7.5 GameState.java File Reference

Classes

- enum org.engine.GameState

  Represents all possible game states.

Packages

- package org.engine

7.6 GameState.java

Go to the documentation of this file.

```
00001 package org.engine;
00006 public enum GameState {
00007     MENU, GAME, GAME_OVER, GAME_OVER_ENTERNAME, LEADERBOARD, TUTORIAL
00008 }
```

7.7 StateChangeListener.java File Reference

Classes

- interface org.engine.StateChangeListener

  Interface to allow state switching in the engine from the states (Observer).

Packages

- package org.engine

7.8 StateChangeListener.java

Go to the documentation of this file.

```
00001 package org.engine;
00002
00007 public interface StateChangeListener {
00008     void changeState(GameState newState);
00009 }
```

7.9 BonusFood.java File Reference

Classes

- class org.objects.food.BonusFood

  Represents bonus food.

Packages

- package org.objects.food

7.10 BonusFood.java

Go to the documentation of this file.

```
00001 package org.objects.food;
00002
00003 import java.awt.*;
00004 import org.utilities.GameConstants;
00005
00013 public class BonusFood extends Food {
00014     private String icon;
00015
00021     public BonusFood() {
00022         super();
00023     }
00024
00030     @Override
00031     public void respawn() {
00032         super.respawn();
00033         randType();
00034     }
00035
00043     @Override
00044     public void draw(Graphics2D frame) {
00045         super.draw(frame);
00046         Point coords = foodLocation.getCoordinates();
00047
00048         Font font = new Font("Public Pixel", Font.BOLD, 11);
00049         frame.setFont(font);
00050
00051         FontMetrics metrics = frame.getFontMetrics(font); // for position calculation
00052         float x = coords.x + GameConstants.CELL_SIZE / 2f - metrics.stringWidth(icon) / 2f;
00053         float y = coords.y + GameConstants.CELL_SIZE / 2f - metrics.getHeight() / 2f +
      metrics.getAscent();
00054
00055         frame.setColor(Color.BLACK);
00056         frame.drawString(icon, x, y);
00057     }
00058
00065     private void randType() {
00066         switch (rand.nextInt(5)) {
00067             case 0 -> {
00068                 this.type = FoodType.SPEEDFOOD;
00069                 this.color = new Color(0xffbf00);
00070                 this.icon = "^";
00071             }
00072             case 1 -> {
00073                 this.type = FoodType.SLOWFOOD;
00074                 this.color = new Color(0x2F38B4);
00075                 this.icon = "v";
00076             }
00077             case 2 -> {
00078                 this.type = FoodType.PLUSFOOD;
00079                 this.color = new Color(0x04B000);
00080                 this.icon = "2";
00081             }
00082             case 3 -> {
00083                 this.type = FoodType.MINUSFOOD;
00084                 this.color = new Color(0xd40000);
00085                 this.icon = "2";
00086             }
00087             case 4 -> {
00088                 this.type = FoodType.CONTROLINVERTER;
00089                 this.color = new Color(0x68009c);
00090                 this.icon = "?";
00091             }
00092         }
00093     }
00094 }
```

7.11 Food.java File Reference

Classes

- class org.objects.food.Food

Represents a food item.

Packages

- package org.objects.food

7.12 Food.java

Go to the documentation of this file.

```
00001 package org.objects.food;
00002
00003 import java.awt.*;
00004 import java.util.Random;
00005
00006 import org.utilities.CellPosition;
00007 import org.utilities.GameConstants;
00008
00009
00014 public class Food {
00015     protected static final int BORDER_SIZE = 2; // thickness of the border of the food item
00016
00017     protected CellPosition foodLocation;
00018     protected Random rand;
00019     protected Color color;
00020     protected FoodType type;
00021
00027     public Food() {
00028         foodLocation = new CellPosition();
00029         rand = new Random();
00030         color = new Color(0x2b331a);
00031         type = FoodType.DEFAULT;
00032         respawn();
00033     }
00034
00039     public void respawn() {
00040         int randX = rand.nextInt(GameConstants.MAX_CELL - GameConstants.MIN_CELL + 1) +
      GameConstants.MIN_CELL;
00041         int randY = rand.nextInt(GameConstants.MAX_CELL - GameConstants.MIN_CELL + 1) +
      GameConstants.MIN_CELL;
00042         foodLocation = new CellPosition(randX, randY);
00043     }
00044
00050     public void draw (Graphics2D frame) {
00051         Point coords = foodLocation.getCoordinates(); // top left coords of the cell
00052         int halfCell = GameConstants.CELL_SIZE / 2;
00053         int[] xPoints, yPoints; // romb point coordinates, clockwise, starting from left (9 o'clock)
      corner
00054
00055         // draw border
00056         frame.setColor(Color.BLACK);
00057         xPoints = new int[]{coords.x - BORDER_SIZE, coords.x + halfCell, coords.x + 2*halfCell +
      BORDER_SIZE, coords.x + halfCell};
00058         yPoints = new int[]{coords.y + halfCell, coords.y - BORDER_SIZE, coords.y + halfCell, coords.y
      + 2*halfCell + BORDER_SIZE};
00059         frame.fillPolygon(xPoints, yPoints,4);
00060
00061         // draw colored middle
00062         frame.setColor(color);
00063         xPoints = new int[]{coords.x, coords.x + halfCell, coords.x + 2*halfCell, coords.x +
      halfCell};
00064         yPoints = new int[]{coords.y + halfCell, coords.y, coords.y + halfCell, coords.y +
      2*halfCell};
00065         frame.fillPolygon(xPoints, yPoints,4);
00066     }
00067
00073     public CellPosition getFoodLocation () {
00074         return this.foodLocation;
00075     }
00076
00082     public FoodType getFoodType() {
00083         return this.type;
00084     }
00085 }
```

7.13 FoodType.java File Reference

Classes

- enum org.objects.food.FoodType

  Represents all possible food types.

Packages

- package org.objects.food

7.14 FoodType.java

Go to the documentation of this file.

```
00001 package org.objects.food;
00002
00007 public enum FoodType {
00008     DEFAULT, SPEEDFOOD, SLOWFOOD, MINUSFOOD, PLUSFOOD, CONTROLINVERTER
00009 }
```

7.15 Obstacle.java File Reference

Classes

- class org.objects.obstacle.Obstacle

  Represents an obstacle.

Packages

- package org.objects.obstacle

7.16 Obstacle.java

Go to the documentation of this file.

```
00001 package org.objects.obstacle;
00002
00003 import org.utilities.CellPosition;
00004 import org.utilities.GameConstants;
00005
00006 import java.awt.*;
00007 import java.util.ArrayList;
00008 import java.util.Random;
00009
00014 public class Obstacle {
00015     private static final int MAX_SIZE = 5; // maximum amt of cells in an obstacle
00016     private static final int PARTICLE_COUNT = 8;
00017     private static final int PARTICLE_SIZE = GameConstants.CELL_SIZE / 5;
00018
00019     private ArrayList<CellPosition> cells;
00020     private Random rand;
00021
00028     public Obstacle(ArrayList<CellPosition> snakePos) {
00029         rand = new Random();
00030         cells = new ArrayList<>();
00031         respawn(snakePos);
```

```
00032     }
00033
00040     public void respawn(ArrayList<CellPosition> snakePos) {
00041         CellPosition startPos = getRandomCell(); // gets a random cell to start the spawning process
00042         if (snakePos.contains(startPos)) return; // prevent spawning if started spawning inside of the
      snake
00043
00044         cells.add(startPos); // add the starting cell to the list of obstacle cells
00045
00046         double growChance = 0.75; // chance for increasing the number of cells
00047         // generate additional cells, up to MAX_SIZE, with diminishing probability
00048         for (int i = 0; i < MAX_SIZE; i++) {
00049             double roll = rand.nextDouble(); // random number to determine if the obstacle should grow
      more
00050
00051             if (roll <= growChance) {
00052                 // grow in a random direction
00053                 ArrayList<CellPosition> viableCells = new ArrayList<>();
00054                 CellPosition prevCell = cells.get(i);
00055
00056                 if (prevCell.x + 1 <= GameConstants.MAX_CELL) // can grow to the right
00057                     viableCells.add(new CellPosition(prevCell.x + 1, prevCell.y));
00058                 if (prevCell.x - 1 >= GameConstants.MIN_CELL) // can grow to the left
00059                     viableCells.add(new CellPosition(prevCell.x - 1, prevCell.y));
00060                 if (prevCell.y + 1 <= GameConstants.MAX_CELL) // can grow down
00061                     viableCells.add(new CellPosition(prevCell.x, prevCell.y + 1));
00062                 if (prevCell.y - 1 >= GameConstants.MIN_CELL) // can grow up
00063                     viableCells.add(new CellPosition(prevCell.x, prevCell.y - 1));
00064
00065                 if (viableCells.isEmpty()) // stop spawning if no viable cells found
00066                     break;
00067
00068                 // choose a random cell out of the possible cells
00069                 CellPosition nextCell = viableCells.get(rand.nextInt(viableCells.size()));
00070                 if (snakePos.contains(nextCell)) // stop spawning if spawned inside the snake
00071                     break;
00072
00073                 cells.add(nextCell);
00074                 growChance = growChance - 0.1; // add non-linearity to the chance
00075             }
00076             else break;
00077         }
00078     }
00079
00085     private CellPosition getRandomCell() {
00086         int randX = rand.nextInt(GameConstants.MAX_CELL - GameConstants.MIN_CELL + 1) +
      GameConstants.MIN_CELL;
00087         int randY = rand.nextInt(GameConstants.MAX_CELL - GameConstants.MIN_CELL + 1) +
      GameConstants.MIN_CELL;
00088         return new CellPosition(randX, randY);
00089     }
00090
00097     public void draw(Graphics2D frame) {
00098         for (CellPosition pos : cells) {
00099             Point p = pos.getCoordinates();
00100
00101             // draw the obstacle
00102             frame.setColor(Color.BLACK);
00103             frame.fillRect(p.x, p.y, GameConstants.CELL_SIZE, GameConstants.CELL_SIZE);
00104
00105             // draw the "noise" in the obstacle
00106             for (int i = 0; i < PARTICLE_COUNT; i++) {
00107                 if (rand.nextFloat() > 0.25)
00108                     frame.setColor(Color.DARK_GRAY);
00109                 else
00110                     frame.setColor(Color.GRAY);
00111
00112                 int x = (int) (rand.nextFloat() * (GameConstants.CELL_SIZE - PARTICLE_SIZE)) + p.x;
00113                 int y = (int) (rand.nextFloat() * (GameConstants.CELL_SIZE - PARTICLE_SIZE)) + p.y;
00114                 frame.fillRect(x, y, PARTICLE_SIZE, PARTICLE_SIZE);
00115             }
00116         }
00117     }
00118
00124     public ArrayList<CellPosition> getCells() {
00125         return cells;
00126     }
00127 }
```

7.17 ObstacleList.java File Reference

Classes

- class org.objects.obstacle.ObstacleList

  Represents a List of all existing obstacles in the game.

Packages

- package org.objects.obstacle

7.18 ObstacleList.java

Go to the documentation of this file.

```
00001 package org.objects.obstacle;
00002
00003 import org.utilities.CellPosition;
00004
00005 import java.util.ArrayList;
00006
00011 public class ObstacleList {
00012     private ArrayList<Obstacle> obstacles;
00013
00018     public ObstacleList() {
00019         obstacles = new ArrayList<>();
00020     }
00021
00027     public ArrayList<CellPosition> getAllCells() {
00028         ArrayList<CellPosition> cells = new ArrayList<>();
00029         for (Obstacle obstacle : obstacles) {
00030             cells.addAll(obstacle.getCells());
00031         }
00032         return cells;
00033     }
00034
00040     public void add(Obstacle obstacle) {
00041         obstacles.add(obstacle);
00042     }
00043
00049     public ArrayList<Obstacle> getObstacles() {
00050         return obstacles;
00051     }
00052 }
```

7.19 Snake.java File Reference

Classes

- class org.objects.Snake

  Represents a snake object.

Packages

- package org.objects

7.20 Snake.java

Go to the documentation of this file.

```
00001 package org.objects;
00002
00003 import org.utilities.CellPosition;
00004 import org.utilities.Direction;
00005 import org.utilities.GameConstants;
00006
00007 import java.awt.*;
00008 import java.util.ArrayList;
00009 import java.util.LinkedList;
00010
00015 public class Snake {
00016     public static final double SPEED = 0.18; // FPS multiplier
00017     public static final int INIT_LEN = 5; // FPS multiplier
00018
00019     private ArrayList<CellPosition> body;
00020     private Direction currentDirection;
00021     private LinkedList<Direction> inputQueue;
00022
00027     public Snake() {
00028         body = new ArrayList<>();
00029         currentDirection = Direction.RIGHT;
00030         inputQueue = new LinkedList<>();
00031
00032         // body elements starting from the middle of the screen and "growing" to the left
00033         int xPos = GameConstants.CELL_COUNT / 2;
00034         for (int i = 0; i < INIT_LEN; i++)
00035             body.add(new CellPosition(xPos--, GameConstants.CELL_COUNT / 2));
00036     }
00037
00043     public ArrayList<CellPosition> getBody() {
00044         return body;
00045     }
00046
00052     private CellPosition calculateNextPos() {
00053         CellPosition currHeadPos = body.get(0);
00054         CellPosition nextPos = null;
00055
00056         switch (currentDirection) {
00057             case UP -> nextPos = new CellPosition(currHeadPos.x, currHeadPos.y - 1);
00058             case DOWN -> nextPos = new CellPosition(currHeadPos.x, currHeadPos.y + 1);
00059             case RIGHT -> nextPos = new CellPosition(currHeadPos.x + 1, currHeadPos.y);
00060             case LEFT -> nextPos = new CellPosition(currHeadPos.x - 1, currHeadPos.y);
00061         }
00062
00063         return nextPos;
00064     }
00065
00072     private boolean doSelfCollision(CellPosition head) {
00073         return body.subList(1, body.size())
00074                 .contains(head);
00075     }
00076
00083     private boolean doBorderCollision(CellPosition head) {
00084         Point nextCoords = head.getCoordinates();
00085         if ((nextCoords.x >= GameConstants.WINDOW_SIZE.x - GameConstants.MARGIN_INNER) ||
      (nextCoords.x < GameConstants.MARGIN_INNER)) {
00086             return true;
00087         }
00088         if ((nextCoords.y >= GameConstants.WINDOW_SIZE.y - GameConstants.MARGIN_INNER) ||
      (nextCoords.y < GameConstants.MARGIN_INNER)) {
00089             return true;
00090         }
00091         return false;
00092     }
00093
00101     public boolean doCollisions() {
00102         CellPosition headPos = body.get(0);
00103         return doSelfCollision(headPos) || doBorderCollision(headPos);
00104     }
00105
00111     public void move() {
00112         if (!inputQueue.isEmpty())
00113             currentDirection = inputQueue.poll(); // change direction if any inputs queued
00114
00115         CellPosition newHeadPos = calculateNextPos();
00116         body.add(0, newHeadPos);
00117     }
00118
00125     public void updateDirection(Direction newDir) {
00126         if (inputQueue.size() < 2 && !isOppositeDir(inputQueue.peekLast(), newDir))
00127             inputQueue.add(newDir);
00128
```

```
00129         if (!inputQueue.isEmpty() && isOppositeDir(inputQueue.peek(), currentDirection))
00130             inputQueue.removeFirst(); // drops the next direction in queue if it is opposite to the
      current direction
00131     }
00132
00140     private boolean isOppositeDir(Direction dir1, Direction dir2) {
00141         return (dir1 == Direction.DOWN && dir2 == Direction.UP) ||
00142                 (dir1 == Direction.UP && dir2 == Direction.DOWN) ||
00143                 (dir1 == Direction.LEFT && dir2 == Direction.RIGHT) ||
00144                 (dir1 == Direction.RIGHT && dir2 == Direction.LEFT);
00145     }
00146
00152     public void draw(Graphics2D frame) {
00153         frame.setColor(new Color(0x2b331a));
00154         for (CellPosition pos : body) {
00155             Point p = pos.getCoordinates();
00156             frame.fillRect(p.x, p.y, GameConstants.CELL_SIZE, GameConstants.CELL_SIZE);
00157         }
00158     }
00159
00166     public boolean checkCollisionWith(CellPosition pos) {
00167         return body.contains(pos);
00168     }
00169
00176     public boolean checkCollisionWith(ArrayList<CellPosition> pos) {
00177         if (pos.isEmpty()) return false;
00178
00179         for (CellPosition p : pos) {
00180             if (body.contains(p)) return true;
00181         }
00182         return false;
00183     }
00184 }
```

7.21 BgPanel.java File Reference

Classes

- class org.panels.BgPanel

  A Panel that represents the game's background to be reused in every game screen.

Packages

- package org.panels

7.22 BgPanel.java

Go to the documentation of this file.

```
00001 package org.panels;
00002
00003 import javax.swing.JPanel;
00004 import java.awt.*;
00005
00006 import static org.utilities.GameConstants.*;
00007
00012 public class BgPanel extends JPanel {
00017     public BgPanel() {
00018         super();
00019         this.setPreferredSize(new Dimension(WINDOW_SIZE.x, WINDOW_SIZE.y));
00020         this.setBackground(new Color(0xA9E000));
00021         this.setDoubleBuffered(true);
00022     }
00023
00031     @Override
00032     public void paintComponent(Graphics g) {
00033         super.paintComponent(g);
00034
00035         Graphics2D frame = (Graphics2D) g;
00036
```

```
00037         // fill background
00038         frame.setColor(new Color(0xA9E000));
00039         frame.fillRect(0, 0, WINDOW_SIZE.x, WINDOW_SIZE.y);
00040
00041         // draw borders
00042         frame.setColor(Color.BLACK);
00043
00044         frame.fillRect(MARGIN_OUTER, MARGIN_OUTER, WINDOW_SIZE.x - 2 * MARGIN_OUTER, BORDER_THC); //
      top border
00045         frame.fillRect(MARGIN_OUTER, MARGIN_OUTER, BORDER_THC, WINDOW_SIZE.y - 2 * MARGIN_OUTER); //
      left border
00046         frame.fillRect(WINDOW_SIZE.x - MARGIN_OUTER - BORDER_THC, MARGIN_OUTER, BORDER_THC,
      WINDOW_SIZE.y - 2 * MARGIN_OUTER); // right border
00047         frame.fillRect(MARGIN_OUTER, WINDOW_SIZE.y - MARGIN_OUTER - BORDER_THC, WINDOW_SIZE.x - 2 *
      MARGIN_OUTER, BORDER_THC); // bottom border
00048
00049         // draw outer border frame
00050         frame.fillRect(0, 0, WINDOW_SIZE.x, BORDER_THC); // top border
00051         frame.fillRect(0, 0, BORDER_THC, WINDOW_SIZE.y); //left border
00052         frame.fillRect(WINDOW_SIZE.x - BORDER_THC, 0, BORDER_THC, WINDOW_SIZE.y); // right border
00053         frame.fillRect(0, WINDOW_SIZE.y - BORDER_THC, WINDOW_SIZE.x, BORDER_THC); // bottom border
00054     }
00055 }
```

7.23 GameOver.java File Reference

Classes

- class org.panels.GameOver

  A panel that represents the game-over screen.

Packages

- package org.panels

7.24 GameOver.java

Go to the documentation of this file.

```
00001 package org.panels;
00002
00003 import org.engine.*;
00004 import org.utilities.GameButton;
00005 import org.utilities.GameConstants;
00006
00007 import javax.swing.*;
00008 import java.awt.*;
00009 import java.awt.event.*;
00010 import java.util.ArrayList;
00011
00016 public class GameOver extends JPanel implements ActionListener, KeyListener, FocusListener {
00017     private final GameButton retryBtn; // Declaring button references
00018     private final GameButton mainMenuBtn;
00019     public BgPanel bg; // BgPanel reference for instantiation
00020     private final ArrayList<GameButton> buttons; //declaring arrayList of Buttons to perform redundant
      button-tasks.
00021
00022     private StateChangeListener stateChanger;
00023
00024     private JLabel scoreText;
00025     private JTextField enterNameField;
00026     private int score;
00027
00038     public GameOver(StateChangeListener listener, int score, boolean isHighScore) {
00039         this.score = score;
00040         stateChanger = listener;
00041
00042         this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS)); // creates a box layout for the panel.
00043         this.setPreferredSize(new Dimension(GameConstants.WINDOW_SIZE.x,
      GameConstants.WINDOW_SIZE.y));
```

```
00044         this.setBackground(Color.decode("#A9E000"));
00045         this.add(Box.createRigidArea(new Dimension(0, 10))); // Gives some space over title
00046         JLabel titleLabel = new JLabel("GAME OVER!", SwingConstants.CENTER);
00047         titleLabel.setForeground(Color.BLACK);
00048         titleLabel.setFont(new Font("Public Pixel", Font.BOLD, 25));
00049         titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
00050         this.add(Box.createRigidArea(new Dimension(0, 5))); // creates a blank area under title for
      visual spacing.
00051         this.add(titleLabel);
00052 
00053         bg = new BgPanel();
00054         retryBtn = new GameButton("Retry"); // Assigning buttons
00055         mainMenuBtn = new GameButton("Main Menu");
00056 
00057         buttons = new ArrayList<>(); // initializing the Button ArrayList.
00058         buttons.add(retryBtn); // adding the existing Button objects to the list.
00059         buttons.add(mainMenuBtn);
00060 
00061         if( isHighScore ){
00062             JLabel newHighScoreLabel = new JLabel("NEW HIGH SCORE!",SwingConstants.CENTER); // A label
      to display "NEW HIGH SCORE!".
00063             newHighScoreLabel.setForeground(Color.BLACK);
00064             newHighScoreLabel.setFont(new Font("Public Pixel", Font.BOLD, 30));
00065             newHighScoreLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
00066             this.add(Box.createRigidArea(new Dimension(0, 100)));  // Add space above the
      newHighScoreLabel.
00067             this.add(newHighScoreLabel); // adds the label to the panel
00068 
00069             scoreText = new JLabel("YOUR SCORE:",SwingConstants.CENTER);
00070             scoreText.setForeground(Color.BLACK);
00071             scoreText.setFont(new Font("Public Pixel", Font.BOLD, 35)); // Sets the font and size
00072             scoreText.setAlignmentX(Component.CENTER_ALIGNMENT);
00073             this.add(Box.createRigidArea(new Dimension(0, 50))); // Adds space above YOUR SCORE label
00074             this.add(scoreText);
00075             this.add(Box.createRigidArea(new Dimension(0, 50))); // Adds visual space between the
      label and score.
00076 
00077             JLabel ScoreLabel = new JLabel(String.valueOf(score),SwingConstants.CENTER); // Adding the
      score as a label, following our design protoType.
00078             ScoreLabel.setForeground(Color.BLACK);
00079             ScoreLabel.setFont(new Font("Public Pixel", Font.BOLD, 35));
00080             ScoreLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
00081             this.add(ScoreLabel);
00082 
00083             JLabel EnterNameLabel = new JLabel("ENTER YOUR NAME:",SwingConstants.CENTER);
00084             EnterNameLabel.setForeground(Color.BLACK);
00085             EnterNameLabel.setFont(new Font("Public Pixel", Font.BOLD, 30));
00086             EnterNameLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
00087             this.add(Box.createRigidArea(new Dimension(0, 170)));
00088             this.add(EnterNameLabel);
00089 
00090             enterNameField = getjTextField(); // Creates a text field
00091             setVisible(true);
00092             this.add(Box.createRigidArea(new Dimension(0,50))); // Add the text field to the panel
      with spacing adjustments.
00093             this.add(enterNameField);
00094             this.add(Box.createRigidArea(new Dimension(0, 50)));
00095 
00096             SwingUtilities.invokeLater(() -> {
00097                 enterNameField.requestFocusInWindow();   // Set the focus on the JTextField
00098             });
00099 
00100 
00101         } else {
00102             this.add(Box.createRigidArea(new Dimension(0, 80)));
00103             scoreText = new JLabel("YOUR SCORE:",SwingConstants.CENTER);
00104             scoreText.setForeground(Color.BLACK);
00105             scoreText.setFont(new Font("Public Pixel", Font.BOLD, 35));
00106             scoreText.setAlignmentX(Component.CENTER_ALIGNMENT);
00107             this.add(Box.createRigidArea(new Dimension(0, 50)));
00108             this.add(scoreText);
00109             this.add(Box.createRigidArea(new Dimension(0, 50)));
00110 
00111             JLabel ScoreLabel = new JLabel(String.valueOf(score),SwingConstants.CENTER);
00112             ScoreLabel.setForeground(Color.BLACK);
00113             ScoreLabel.setFont(new Font("Public Pixel", Font.BOLD, 35));
00114             ScoreLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
00115             this.add(ScoreLabel);
00116 
00117             this.add(Box.createRigidArea(new Dimension(0, 160)));  // Adds space above the new score
      label
00118 
00119             for (GameButton button : buttons) {
00120                 Font newButtonFont = new Font("Public Pixel", Font.BOLD, 35);
00121                 button.setFont(newButtonFont);
00122                 button.setPreferredSize(new Dimension(700, 120));
00123                 button.setMaximumSize(new Dimension(700, 120));
```

```
00124                 this.add(button); // Adds the buttons to the panel.
00125             }
00126 
00127             retryBtn.setActionCommand("retry");
00128             retryBtn.addActionListener( this);
00129 
00130             mainMenuBtn.setActionCommand("menu");
00131             mainMenuBtn.addActionListener(this);
00132 
00133         }
00134     }
00135 
00141     private JTextField getjTextField() {
00142         JTextField enterNameField = new JTextField(SwingConstants.CENTER); // Declaring a private
      static method that returns a JTextField instance, with the initial text (___), and centers the text
      within the field.
00143         enterNameField.add(Box.createRigidArea(new Dimension(0,100))); //
00144         enterNameField.setBorder(BorderFactory.createEmptyBorder()); // Sets an empty border around
      the text field.
00145         enterNameField.setForeground(Color.BLACK); // Sets the text color of the field to black.
00146         enterNameField.setBackground(Color.decode("#A9E000"));
00147         enterNameField.setFont(new Font("Public Pixel", Font.BOLD,30));
00148         enterNameField.setAlignmentX(Component.CENTER_ALIGNMENT);
00149         enterNameField.setMaximumSize(new Dimension(100, 50));
00150         enterNameField.addFocusListener(this);  //  Allows the user to start typing without manually
      removing the initial placeholder text.
00151         enterNameField.addKeyListener(this);
00152 
00153         return enterNameField;
00154     }
00155 
00161     @Override
00162     protected void paintComponent(Graphics graphics) {
00163         super.paintComponent(graphics);
00164         bg.paintComponent(graphics);
00165     }
00166 
00172     @Override
00173     public void actionPerformed(ActionEvent event ) {
00174         String actionCommand = event.getActionCommand();
00175 
00176         if ("retry".equals(actionCommand)) {
00177             stateChanger.changeState(GameState.GAME);
00178         } else if ("menu".equals(actionCommand)) {
00179             stateChanger.changeState(GameState.MENU);
00180         }
00181     }
00182 
00189     @Override
00190     public void keyTyped(KeyEvent e) {
00191         char c = e.getKeyChar();
00192         if (enterNameField.getText().length() >= 3 ||(!Character.isLetter(c) &&
      !Character.isWhitespace(c)) )
00193             e.consume();
00194     }
00195 
00201     @Override
00202     public void keyPressed(KeyEvent e) {
00203         if(e.getKeyCode() == KeyEvent.VK_ENTER && !enterNameField.getText().isEmpty()){
00204             Leaderboard.createPlayer(enterNameField.getText(), score);
00205             stateChanger.changeState(GameState.LEADERBOARD);
00206         }
00207     }
00208 
00209     @Override
00210     public void keyReleased(KeyEvent e) {}
00211 
00217     @Override
00218     public void focusGained(FocusEvent e) {
00219         if( enterNameField.getText().isBlank() ){
00220             enterNameField.setText("");
00221         }
00222     }
00223 
00224     @Override
00225     public void focusLost(FocusEvent e) {}
00226 }
```

7.25 GamePanel.java File Reference

Classes

- class org.panels.GamePanel

Represents the gameplay state.

Packages

- package org.panels

7.26 GamePanel.java

Go to the documentation of this file.

```
00001 package org.panels;
00002
00003 import org.objects.*;
00004 import org.objects.food.*;
00005 import org.objects.obstacle.*;
00006 import org.engine.*;
00007 import org.utilities.CellPosition;
00008 import org.utilities.Direction;
00009 import org.utilities.GameConstants;
00010 import org.utilities.Player;
00011
00012 import javax.swing.*;
00013 import java.awt.*;
00014 import java.awt.event.KeyEvent;
00015 import java.awt.event.KeyListener;
00016
00017 import java.util.ArrayList;
00018 import java.util.Iterator;
00019 import java.util.Random;
00020
00021
00027 public class GamePanel extends JPanel implements KeyListener {
00028     private BgPanel bg;
00029     private Snake snake;
00030     private ArrayList<Food> food;
00031     private Random rand;
00032     private ObstacleList obstacles;
00033
00034     private int score;
00035     private final Timer gameLoop;
00036     private long startTime;
00037     String currentEffectLabel;
00038     private boolean fastMode, slowMode, keyInverter;
00039     private StateChangeListener stateChanger;
00040
00047     public GamePanel(StateChangeListener listener) {
00048         super();
00049         this.setPreferredSize(new Dimension(GameConstants.WINDOW_SIZE.x,
      GameConstants.WINDOW_SIZE.y));
00050         this.setBackground(Color.BLACK);
00051         this.setDoubleBuffered(true);
00052         this.setFocusable(true);
00053
00054         rand = new Random();
00055         bg = new BgPanel();
00056         snake = new Snake();
00057
00058         stateChanger = listener;
00059         food = new ArrayList<>();
00060         food.add(new Food());
00061         obstacles = new ObstacleList();
00062         score = 0;
00063         startTime = 0;
00064         fastMode = false;
00065         slowMode = false;
00066
00067         int delay = (int) (1000 / (GameConstants.FPS * Snake.SPEED));
00068         gameLoop = new Timer(delay, e -> { // GAME LOOP, runs every 1/60*SPEED -th of a second
00069
00070             //1000/(int)(FPS * objects.Snake.SPEED)
00071             update();
00072             repaint(); // calls paintComponent()
00073
00074         });
00075     }
00076
00082     public void update() {
00083         updateEffects();
```

```
00084         snake.move(); // add a new "head" based on the movement direction
00085
00086         // snake collisions (self and borders)
00087         if (snake.doCollisions()) {
00088             stopGame();
00089             return;
00090         }
00091
00092         // collision with obstacles
00093         if (snake.checkCollisionWith(obstacles.getAllCells())) {
00094             stopGame();
00095             return;
00096         }
00097
00098         // collision with food
00099         boolean incLength = doFoodCollisions();
00100         // do not remove tail for a length-increase effect
00101         if(!incLength) {
00102             snake.getBody().remove(snake.getBody().size() - 1); // remove the tail to complete
    movement
00103         }
00104     }
00105
00111     private void adjustSnakeSpeed(double speedMultiplier) {
00112         int delay = (int) (1000 / (GameConstants.FPS * Snake.SPEED * speedMultiplier));
00113         gameLoop.setDelay(delay);
00114     }
00115
00120     private void updateEffects() {
00121         if (fastMode || slowMode || keyInverter) {
00122             // check if effect time expired
00123             if (System.currentTimeMillis() - startTime > GameConstants.EFFECT_DURATION) {
00124                 fastMode = false;
00125                 slowMode = false;
00126                 keyInverter = false;
00127                 adjustSnakeSpeed(1); // Set the speed back to normal
00128                 currentEffectLabel = ""; // remove the label
00129             }
00130         }
00131     }
00132
00139     private boolean doFoodCollisions() {
00140         boolean eaten = false;
00141         Iterator<Food> it = food.iterator();
00142         ArrayList<Food> newFood = new ArrayList<>();
00143
00144         while (it.hasNext()) {
00145             Food foodItem = it.next();
00146             if (snake.checkCollisionWith(foodItem.getFoodLocation())) {
00147                 it.remove(); // remove consumed food from list
00148                 eaten = true;
00149                 applyFoodEffect(foodItem.getFoodType());
00150
00151                 // always spawn new default food as soon as the previous one is eaten
00152                 if(foodItem.getFoodType() == FoodType.DEFAULT)
00153                     newFood.add(generateNewFoodItem(false));
00154
00155                 // 33% to spawn new bonus food in a valid position, up to 1 normal and 2 bonus
00156                 if (food.size() < 2 && rand.nextFloat() <= 0.33)
00157                     newFood.add(generateNewFoodItem(true));
00158
00159                 // spawn new obstacle every 5th time food is eaten
00160                 if (score % 5 == 0)
00161                     obstacles.add(new Obstacle(snake.getBody()));
00162             }
00163         }
00164
00165         food.addAll(newFood); // add all generated food from the temporary storage to the actual list
00166         return eaten;
00167     }
00168
00176     private void applyFoodEffect(FoodType foodType) {
00177         switch (foodType) {
00178             case DEFAULT -> {
00179                 score++;
00180             }
00181             case SPEEDFOOD -> {
00182                 fastMode = true;
00183                 adjustSnakeSpeed(2);
00184                 currentEffectLabel = "SPED UP!";
00185                 startTime = System.currentTimeMillis();
00186             }
00187             case SLOWFOOD -> {
00188                 slowMode = true;
00189                 adjustSnakeSpeed(0.5);
00190                 currentEffectLabel = "SLOWED!";
00191                 startTime = System.currentTimeMillis();
```

```
00192             }
00193             case PLUSFOOD -> {
00194                 score += 2;
00195             }
00196             case MINUSFOOD -> {
00197                 score -= 2;
00198                 if (score < 0) score = 0;
00199             }
00200             case CONTROLINVERTER -> {
00201                 keyInverter = true;
00202                 startTime = System.currentTimeMillis();
00203                 adjustSnakeSpeed(0.75);
00204                 currentEffectLabel = "CONFUSED!";
00205             }
00206         }
00207     }
00208
00215     private Food generateNewFoodItem(boolean isBonus) {
00216         CellPosition newFoodPos;
00217         Food newFood;
00218
00219         // respawn food until it's in a valid position
00220         do {
00221             if (!isBonus) newFood = new Food();
00222             else newFood = new BonusFood();
00223             newFoodPos = newFood.getFoodLocation();
00224         } while (snake.checkCollisionWith(newFoodPos) ||
    obstacles.getAllCells().contains(newFoodPos));
00225
00226         return newFood;
00227     }
00228
00234     public int getScore() {
00235         return this.score;
00236     }
00237
00245     @Override
00246     public void paintComponent(Graphics g) {
00247         super.paintComponent(g);
00248         bg.paintComponent(g); // draw background first
00249
00250         Graphics2D frame = (Graphics2D) g; // frame for drawing 2d graphics
00251
00252         for (Food foodItem : food)
00253             foodItem.draw(frame);
00254         for (Obstacle obstacle : obstacles.getObstacles())
00255             obstacle.draw(frame);
00256
00257         g.setColor(Color.BLACK);
00258         g.setFont(new Font("Public Pixel", Font.PLAIN,20));
00259         g.drawString(String.format("%03d", score), 65 , GameConstants.WINDOW_SIZE.y - 760);
00260
00261         if (currentEffectLabel != null && !currentEffectLabel.isEmpty()) {
00262             frame.setColor(Color.BLACK);
00263             frame.setFont(new Font("Public Pixel", Font.BOLD, 20));
00264
00265             int x = (getWidth() - frame.getFontMetrics().stringWidth(currentEffectLabel)) / 2;
00266             int y = getHeight() - frame.getFontMetrics().getHeight() - 2; // centers the effect label
    2 pixels from the bottom
00267
00268             frame.drawString(currentEffectLabel, x, y);
00269         }
00270
00271         snake.draw(frame);
00272         frame.dispose();
00273     }
00274
00279     public void startGame() {
00280         gameLoop.start();
00281     }
00282
00287     public void stopGame() {
00288         gameLoop.stop();
00289
00290         Player tempPlayer = new Player("", score);
00291
00292         if (score > 0 && Leaderboard.isTopTen(tempPlayer)) {
00293             stateChanger.changeState(GameState.GAME_OVER_ENTERNAME);
00294         }
00295         else {
00296             stateChanger.changeState(GameState.GAME_OVER);
00297         }
00298     }
00299
00300     @Override
00301     public void keyTyped(KeyEvent e) {}
00302
```

```
00309     @Override
00310     public void keyPressed(KeyEvent e) {
00311         int code = e.getKeyCode();
00312 
00313         if (code == KeyEvent.VK_UP) {
00314             if(keyInverter) {
00315                 snake.updateDirection(Direction.DOWN);
00316             }
00317             else {
00318                 snake.updateDirection(Direction.UP);
00319             }
00320         }
00321         if (code == KeyEvent.VK_DOWN) {
00322             if(keyInverter) {
00323                 snake.updateDirection(Direction.UP);
00324             }
00325             else {
00326                 snake.updateDirection(Direction.DOWN);
00327             }
00328         }
00329         if (code == KeyEvent.VK_LEFT) {
00330             if(keyInverter) {
00331                 snake.updateDirection(Direction.RIGHT);
00332             }
00333             else {
00334                 snake.updateDirection(Direction.LEFT);
00335             }
00336         }
00337         if (code == KeyEvent.VK_RIGHT) {
00338             if(keyInverter) {
00339                 snake.updateDirection(Direction.LEFT);
00340             }
00341             else {
00342                 snake.updateDirection(Direction.RIGHT);
00343             }
00344         }
00345     }
00346 
00347     @Override
00348     public void keyReleased(KeyEvent e) {
00349     }
00350 }
```

7.27 Leaderboard.java File Reference

Classes

- class org.panels.Leaderboard

  Panel representing the Leaderboard screen.

Packages

- package org.panels

7.28 Leaderboard.java

Go to the documentation of this file.

```
00001 package org.panels;
00002 
00003 import org.json.simple.JSONObject;
00004 import org.json.simple.parser.JSONParser;
00005 
00006 import javax.swing.*;
00007 import java.awt.*;
00008 import java.awt.event.ActionEvent;
00009 import java.awt.event.ActionListener;
00010 
00011 import java.io.File;
00012 import java.io.FileReader;
```

```
00013 import java.io.FileWriter;
00014 import java.io.IOException;
00015
00016 import java.util.ArrayList;
00017 import java.util.Collections;
00018
00019 import org.utilities.Player;
00020 import org.utilities.GameButton;
00021 import org.utilities.GameConstants;
00022 import org.engine.*;
00023
00028 public class Leaderboard extends JPanel implements ActionListener {
00029     private DefaultListModel<String> listItems;
00030     private JList<String> lbList;
00031     private GameButton mainMenuBtn;     // a Button to go back to the main menu
00032     private BgPanel bg;
00033
00034     private static ArrayList<Player> playerList = new ArrayList<>();
00035
00036     private StateChangeListener stateChanger;
00037
00045     public Leaderboard(StateChangeListener listener){
00046         bg = new BgPanel();
00047         // creating custom font for the game
00048         this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS)); //creates a box layout for the panel.
00049         this.setPreferredSize(new Dimension(GameConstants.WINDOW_SIZE.x,
      GameConstants.WINDOW_SIZE.y));
00050         this.setBackground(Color.decode("#A9E000")); // sets the color to the nokia snake green
      background color.
00051
00052         JLabel titleLabel = new JLabel("Leaderboard", SwingConstants.CENTER); //creates the title
      "snake evolution" for the menu.
00053         titleLabel.setForeground(Color.BLACK); //colors it black.
00054         titleLabel.setFont(new Font("Public Pixel", Font.BOLD,25));
00055         titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT); //centers the text.
00056
00057         // defining layout of list
00058         listItems = new DefaultListModel<>(); // A list that will hold players names and score
00059         lbList = new JList<>(listItems); // a list that will define the layout ie color, size, etc
00060         lbList.setBackground(Color.decode("#A9E000"));
00061         lbList.setForeground(Color.BLACK);
00062         lbList.setFont(new Font("Public Pixel", Font.BOLD,25));
00063
00064         this.add(Box.createRigidArea(new Dimension(0, 10)));//creates a blank area above title for
      visual spacing.
00065         this.add(titleLabel); // adds title to panel.
00066         this.add(Box.createRigidArea(new Dimension(0, 50)));// drawing blank area above JList
00067         this.add(lbList);
00068
00069         mainMenuBtn = new GameButton("Main Menu");
00070         mainMenuBtn.setFocusable(true);
00071         mainMenuBtn.setActionCommand("MENU");
00072
00073         this.add(Box.createRigidArea(new Dimension(0, 250))); // adds empty area before MM button
00074         this.add(mainMenuBtn);
00075         mainMenuBtn.addActionListener(this);
00076         stateChanger = listener;
00077
00078         readToList();
00079     }
00080
00085     @Override
00086     public void paintComponent(Graphics g) {
00087         super.paintComponent(g);
00088         bg.paintComponent(g);// Drawing black border on the frame
00089     }
00090
00096     @Override
00097     public void actionPerformed(ActionEvent event) {
00098         String actionCommand = event.getActionCommand();
00099
00100         if ("MENU".equals(actionCommand)) {
00101             stateChanger.changeState(GameState.MENU); // switches to main menu.
00102         }
00103     }
00104
00110     private static ArrayList<Player> readFromFile() {
00111         JSONParser parser = new JSONParser();
00112         ArrayList<Player> players = new ArrayList<>();
00113         File file = new File("assets/Top10Scores.json");
00114
00115         try {
00116             if (!file.exists()) {
00117                 JSONObject jsonObj = new JSONObject();
00118                 FileWriter writer = new FileWriter(file);
00119                 writer.write(jsonObj.toJSONString());
00120                 writer.close();
```

```
00121             } else {
00122                 FileReader reader = new FileReader(file); // Creating reader to read data from json
      file
00123                 JSONObject readJsonObj = (JSONObject) parser.parse(reader); // parsing data from json
      file to string
00124
00125                 for (Object PlayersData : readJsonObj.keySet()) {
00126                     String playerName = (String) PlayersData;
00127                     long playerScore = (long) readJsonObj.get(PlayersData);
00128                     Player player = new Player(playerName, playerScore);
00129                     players.add(player);
00130                 }
00131
00132                 // sorting players from high to low scores
00133                 Collections.sort(players);
00134             }
00135         }
00136         catch (Exception e) {
00137             throw new RuntimeException("Error while reading file!\n" + e.getMessage());
00138         }
00139
00140         return players;
00141     }
00142
00148     private void readToList() {
00149         ArrayList<Player> top10Scorers = readFromFile();
00150
00151         // filling the list with top 10 players names and scores
00152         int playerIndex = 1;
00153         for (Player player : top10Scorers) {
00154             if (playerIndex < 10)
00155                 listItems.addElement(playerIndex + " . " + player.getNamesAndScores());
00156             else if (playerIndex == 10)
00157                 listItems.addElement(playerIndex + ". " + player.getNamesAndScores());
00158             else
00159                 break; // stop iterating after 10th player
00160             playerIndex++;
00161         }
00162     }
00163
00170     public static void createPlayer(String name, long score){
00171         ArrayList<Player> currentPlayers = readFromFile();
00172
00173         FileWriter writer;
00174         JSONObject jsonObj = new JSONObject();
00175
00176         playerList.addAll(currentPlayers);
00177         Player newPlayer = new Player(name, score);
00178         playerList.add(newPlayer);
00179
00180         // Storing top 10 players' names and scores in json file
00181         try {
00182             writer = new FileWriter("assets/Top10Scores.json");
00183             for (Player player : playerList) {
00184                 if (jsonObj.containsKey(player.getName()) && (long) jsonObj.get(player.getName()) >
      player.getScore()) {
00185                     continue; // do not put a lower score if an entry with the same name exists
00186                 }
00187                 jsonObj.put(player.getName(), player.getScore());
00188             }
00189             writer.write(jsonObj.toJSONString());
00190             writer.close();
00191
00192         } catch (IOException e) {
00193             throw new RuntimeException(e);
00194         }
00195     }
00196
00204     public static boolean isTopTen(Player playerInTop10){
00205         ArrayList<Player> players = readFromFile();
00206         if (players.size() < 10) return true;
00207         return playerInTop10.getScore() > players.get(9).getScore();
00208     }
00209 }
```

7.29 MainMenu.java File Reference

Classes

- class org.panels.MainMenu

  A panel representing the main menu.

Packages

- package org.panels

7.30 MainMenu.java

Go to the documentation of this file.

```
00001 package org.panels;
00002
00003 import org.engine.*;
00004 import org.utilities.GameButton;
00005 import org.utilities.GameConstants;
00006
00007 import javax.swing.*;
00008 import java.awt.*;
00009 import java.awt.event.ActionEvent;
00010 import java.awt.event.ActionListener;
00011 import java.util.ArrayList;
00012
00013
00018 public class MainMenu extends JPanel implements ActionListener { // the mainMenu class javas JPanel &
      implements ActionListener interface.
00019     private GameButton startBtn; // Declaring utilities.Button references
00020
00021     private GameButton tutorialBtn;
00022     private GameButton leaderboardBtn;
00023     private GameButton exitBtn;
00024     public BgPanel bg; // BgPanel reference for instantiation
00025
00026     private ArrayList<GameButton> buttons; //declaring arrayList of Buttons to perform redundant
    button-tasks.
00027
00028     private StateChangeListener stateChanger; // reference to state changer instance.
00029
00035     public MainMenu(StateChangeListener listener) {
00036         bg = new BgPanel();
00037         this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS)); //creates a box layout for the panel.
00038         this.setPreferredSize(new Dimension(GameConstants.WINDOW_SIZE.x,
    GameConstants.WINDOW_SIZE.y));
00039         this.setBackground(Color.decode("#A9E000")); // sets the color to the nokia snake green
    background color.
00040
00041         JLabel titleLabel = new JLabel("Snake Evolution", SwingConstants.CENTER); //creates the title
    "snake evolution" for the menu.
00042         titleLabel.setForeground(Color.BLACK); //colors it black.
00043         titleLabel.setFont(new Font("Public Pixel", Font.BOLD, 25)); //changes font and size.
00044         titleLabel.setAlignmentX(Component.CENTER_ALIGNMENT); //centers the text.
00045         this.add(Box.createRigidArea(new Dimension(0, 15))); //creates a blank area above title for
    visual spacing.
00046         this.add(titleLabel); // adds title to panel.
00047         this.add(Box.createRigidArea(new Dimension(0, 50)));
00048
00049         startBtn = new GameButton("Start"); //assigning buttons.
00050         leaderboardBtn = new GameButton("Leaderboard");
00051         tutorialBtn = new GameButton("Tutorial");
00052         exitBtn = new GameButton("Exit");
00053
00054         buttons = new ArrayList<>();// initializing the utilities.Button ArrayList.
00055         buttons.add(startBtn);// adding the existing utilities.Button objects to the list.
00056         buttons.add(tutorialBtn);
00057         buttons.add(leaderboardBtn);
00058         buttons.add(exitBtn);
00059
00060         for (GameButton button : buttons) {
00061             this.add(button); // add the buttons to the panel
00062             button.setActionCommand(button.getText()); //sets action command for the button that is
    the same as its name
00063             button.addActionListener(this); //adds listener to register button interaction
00064         }
00065
00066         stateChanger = listener;
00067     }
00068
00074     @Override
00075     public void paintComponent(Graphics g) {
00076         super.paintComponent(g);
00077         bg.paintComponent(g);
00078     }
00079
00085     @Override
```

```
00086     public void actionPerformed(ActionEvent event) { // logic for when buttons are clicked.
00087         String actionCommand = event.getActionCommand();
00088 
00089         if ("Start".equals(actionCommand)) {
00090             stateChanger.changeState(GameState.GAME); // switches to state GAME.
00091         }else if ("Tutorial".equals(actionCommand)){
00092             stateChanger.changeState(GameState.TUTORIAL); // switches to state TUTORIAL.
00093         } else if ("Leaderboard".equals(actionCommand)) {
00094             stateChanger.changeState(GameState.LEADERBOARD); // switches to state LEADERBOARD.
00095         } else if ("Exit".equals(actionCommand)) {
00096             System.exit(0); // terminates the program.
00097         }
00098     }
00099 }
00100 
00101 
00102 
00103 
00104 
00105 
00106 
```

7.31 Tutorial.java File Reference

Classes

- class org.panels.Tutorial

  Represents a tutorial screen.

Packages

- package org.panels

7.32 Tutorial.java

Go to the documentation of this file.

```
00001 package org.panels;
00002 
00003 import org.engine.GameState;
00004 import org.engine.StateChangeListener;
00005 import org.utilities.GameButton;
00006 import org.utilities.GameConstants;
00007 
00008 import javax.imageio.ImageIO;
00009 import javax.swing.*;
00010 import java.awt.*;
00011 import java.awt.event.ActionEvent;
00012 import java.awt.event.ActionListener;
00013 import java.awt.image.BufferedImage;
00014 import java.io.File;
00015 import java.io.IOException;
00016 
00021 public class Tutorial extends JPanel implements ActionListener {
00022     private GameButton menuBtn;
00023 
00024     private BufferedImage tutorialPic;
00025     private StateChangeListener stateChanger;
00026 
00033     public Tutorial(StateChangeListener listener) {
00034         this.setLayout(new BoxLayout(this, BoxLayout.Y_AXIS)); //creates a box layout for the panel.
00035         this.setPreferredSize(new Dimension(GameConstants.WINDOW_SIZE.x,
      GameConstants.WINDOW_SIZE.y));
00036         this.setBackground(Color.decode("#A9E000")); // sets the color to the nokia snake green
      background color.
00037 
00038         try {
00039             tutorialPic = ImageIO.read(new File("assets/HowToPlaySnake.png"));
00040         } catch (IOException e) {
00041             throw new RuntimeException(e);
```

```
00042         }
00043         menuBtn = new GameButton("Main Menu");
00044         menuBtn.addActionListener(this);
00045         menuBtn.setActionCommand(menuBtn.getText());
00046         this.add(Box.createRigidArea(new Dimension(0, 650))); // adds empty area before MM button
00047         this.add(menuBtn);
00048 
00049         stateChanger = listener;
00050     }
00051 
00052     @Override
00053     public void paintComponent(Graphics g) {
00054         super.paintComponent(g);
00055         g.drawImage(tutorialPic,0,0,null);
00056     }
00057 
00058     @Override
00059     public void actionPerformed(ActionEvent event) { // logic for when buttons are clicked.
00060         String actionCommand = event.getActionCommand();
00061 
00062         if ("Main Menu".equals(actionCommand)) {
00063             stateChanger.changeState(GameState.MENU); // switches to state MENU.
00064         }
00065     }
00066 }
```

7.33 CellPosition.java File Reference

Classes

- class org.utilities.CellPosition

  Represents the position on screen in cell-system.

Packages

- package org.utilities

7.34 CellPosition.java

Go to the documentation of this file.

```
00001 package org.utilities;
00002 
00003 import java.awt.*;
00004 
00005 
00010 public class CellPosition extends Point {
00011     // x,y - inherited
00016     public CellPosition(){
00017         super(0, 0);
00018     }
00019 
00026     public CellPosition(int initCellX, int initCellY) {
00027         x = initCellX;
00028         y = initCellY;
00029     }
00030 
00036     public Point getCoordinates() { // returns coordinates of the current cell
00037         return new Point(x * GameConstants.CELL_SIZE, y * GameConstants.CELL_SIZE);
00038     }
00039 
00046     @Override
00047     public boolean equals(Object object) {
00048         if (object == null) return false;
00049         if (!(object instanceof CellPosition other)) return false;
00050 
00051         return (this.x == other.x) &&
00052                 (this.y == other.y);
00053     }
00054 }
```

7.35 Direction.java File Reference

Classes

- enum org.utilities.Direction

  Describes all possible directions.

Packages

- package org.utilities

7.36 Direction.java

Go to the documentation of this file.

```
00001 package org.utilities;
00002
00007 public enum Direction {
00008     UP, DOWN, LEFT, RIGHT;
00009 }
```

7.37 GameButton.java File Reference

Classes

- class org.utilities.GameButton

  A JButton that conforms to the specified design.

Packages

- package org.utilities

7.38 GameButton.java

Go to the documentation of this file.

```
00001 package org.utilities;
00002
00003 import javax.swing.*;
00004 import java.awt.*;
00005 import java.awt.event.MouseEvent;
00006 import java.awt.event.MouseListener;
00007
00012 public class GameButton extends JButton implements MouseListener {  // extends the JButton class and
      uses the mouseListener interface.
00013     private final String standardText; // declaring variable to store the name of the button as
      standard text.
00014
00020     public GameButton(String standardText) { // constructor that takes the button name as parameter.
00021         super(standardText);
00022         this.standardText = standardText;
00023         editButton(this); // calls the editButton function when creating a button.
00024         setFocusable(true); // calling setFocusable for every button just in case.
00025         this.addMouseListener(this); // adding a mouseListener to the button upon creation.
00026     }
00027
00033     public void editButton(GameButton button) { // function to apply desired button design
```

```
00034         Font buttonFont = new Font("Public Pixel", Font.BOLD, 50); // instantiating a Font object to
      use as standard font.
00035         this.setFont(buttonFont); // calling setFont on button, using my buttonFont as parameter.
00036         this.setBorderPainted(false); // making borders of button disappear and just display the text.
00037         this.setForeground(Color.BLACK); //setting button text to black.
00038         this.setContentAreaFilled(false); // making the button transparent.
00039         this.setBorder(BorderFactory.createEmptyBorder()); // removes the default JButton borders.
00040         this.setPreferredSize(new Dimension(750, 150)); // setting the size of the buttons to fit the
      text.
00041         this.setMaximumSize(new Dimension(800, 150)); // sets maximum size to same as preferred size
      for consistency.
00042         this.setAlignmentX(Component.CENTER_ALIGNMENT); // centers the buttons in the window.
00043     }
00044
00051     public void onHover(boolean hovering) { // function for mouse hovering.
00052         if (hovering) {
00053             this.setText("<" + standardText + ">" ); // adds the "< >" to the button text upon
      hovering on it.
00054         } else {
00055             this.setText(standardText); // restores the text to only display button text when not
      hovering.
00056         }
00057     }
00058
00059
00060
00061     @Override
00062     public void mouseClicked(MouseEvent e) {} //mandatory part of mouseListener interface, but not
      used.
00063
00064     @Override
00065     public void mousePressed(MouseEvent e) {} //mandatory part of mouseListener interface, but not
      used.
00066
00067     @Override
00068     public void mouseReleased(MouseEvent e) {} //mandatory part of mouseListener interface, but not
      used.
00069
00075     @Override
00076     public void mouseEntered(MouseEvent e) {
00077         this.onHover(true);
00078     }
00079
00085     @Override
00086     public void mouseExited(MouseEvent e) {
00087         this.onHover(false);
00088     }
00089 }
00090
```

7.39 GameConstants.java File Reference

Classes

- interface org.utilities.GameConstants

  Defines constants used in the game.

Packages

- package org.utilities

7.40 GameConstants.java

Go to the documentation of this file.

```
00001 package org.utilities;
00002
00003 import java.awt.Point;
00004
00005
```

```
00010 public interface GameConstants {
00011     // game window constants
00015     static final Point WINDOW_SIZE = new Point(800, 800);
00016
00020     static final int FPS = 60;
00021
00025     static final int CELL_COUNT = 40;
00026
00030     static final int CELL_SIZE = WINDOW_SIZE.x / CELL_COUNT;
00031
00035     static final int EFFECT_DURATION = 8000;
00036
00037     // background constants
00041     static final int BORDER_THC = 5;
00042
00046     static final int MARGIN_CELLS = 3;
00047
00052     static final int MARGIN_INNER = CELL_SIZE * MARGIN_CELLS;
00053
00057     static final int MARGIN_OUTER = MARGIN_INNER - BORDER_THC;
00058
00062     static final int MIN_CELL = MARGIN_CELLS;
00063
00067     static final int MAX_CELL = CELL_COUNT - MARGIN_CELLS - 1;
00068 }
```

7.41 Player.java File Reference

Classes

- class org.utilities.Player

  Represents a player that can be added to the leaderboard.

Packages

- package org.utilities

7.42 Player.java

Go to the documentation of this file.

```
00001 package org.utilities;
00002
00003
00008 public class Player implements Comparable<Player> {
00009
00010     private String name;
00011     private long score;
00012
00019     public Player(String name, long score){
00020         this.name = name;
00021         this.score = score;
00022     }
00023
00029     public String getName(){
00030         return this.name;
00031     }
00032
00038     public long getScore(){
00039         return this.score;
00040     }
00041
00047     public String getNamesAndScores(){
00048         String truncatedName = this.name;
00049         if (this.name.length() > 3) {
00050             truncatedName = this.name.substring(0, 3);
00051         }
00052         return String.format("%-4S--------------%2s", truncatedName, this.score);
00053     }
00054
```

```
00061     @Override
00062     public boolean equals(Object o) {
00063         if (o == this) return true;
00064         if (o == null) return false;
00065         if (!(o instanceof Player)) return false;
00066 
00067         Player other = (Player) o;
00068         return this.name.equals(other.getName()) &&
00069                 this.score == other.getScore();
00070     }
00071 
00079     @Override
00080     public int compareTo(Player other) {
00081         if (this.score < other.score) return 1;
00082         if (this.score > other.score) return -1;
00083         return this.name.compareTo(other.getName());
00084     }
00085 }
```

Index