

**Design Document**

*Requirements*:

* One initial screen welcoming the player.
* A level screen indicating to the player what is needed to progress to the next level.
* A progress screen informing the player if they have succeeded or failed.
* A screen that plays the game.
* Balls should move around the screen at random speeds.
* The first ball should be exploded by the player.
* Any other ball that hits that ball should explode.
* A scoring mechanism to hold the score for each level played.
* A total score so the total score can be saved if the player finishes the game.
* A mechanism that scores the game according to a chain of exploding balls.
* A secret level option should be available to the player.
* Music should play when balls collide and after each level*.*

*Design Process:*

Single class that extends JFrame can hold all the screens throughout the game. According to the game flow this JFrames’ container can be used to add and remove panels to the container thus showing what the player sees. The various screens all extend JPanel thus we can pass the current screen and the next screen to that method.

A ball class can be created to create ball objects. A data structure in the game can be used to hold the ball objects. Methods for collisions and ball movements can be defined here.

A game thread class shall form the bulk of the game. Here we draw balls to the screen and allow the player to interact with the game. Multiple threads can be used to allow the game to flow nicely. One thread can be used to update the balls moving around the JPanel while another can be used to check for collisions after each update. These threads can synchronize access to these two methods by using a lock object and using wait notify. The balls first, update then the thread for collision detecting gets the lock after being notified by the thread running the update. Each time a collision happens the exploding ball must explode and then wait for a few seconds before vanishing. This can be done by just using another class that runs its own thread that can deal with this requirement.

Exploding balls must also change their transparency, this can be done in the paint method using an alpha composite object.

The flow of the game is controlled using mouse and action listeners. These effectively are used to set the container to add and remove the required JPanels.

Each thread dies when the exploding ball list is emptied as this signals the end of the current level. A volatile bool (avoid caching) is set to false.

If the last level is completed the score is saved to a file to meet the high score requirement.

Music and images are used to indicate to the player how they performed after each level.

The secret menu is achieved through a key listener that will give the player a choice to proceed.

*Class Diagram:*

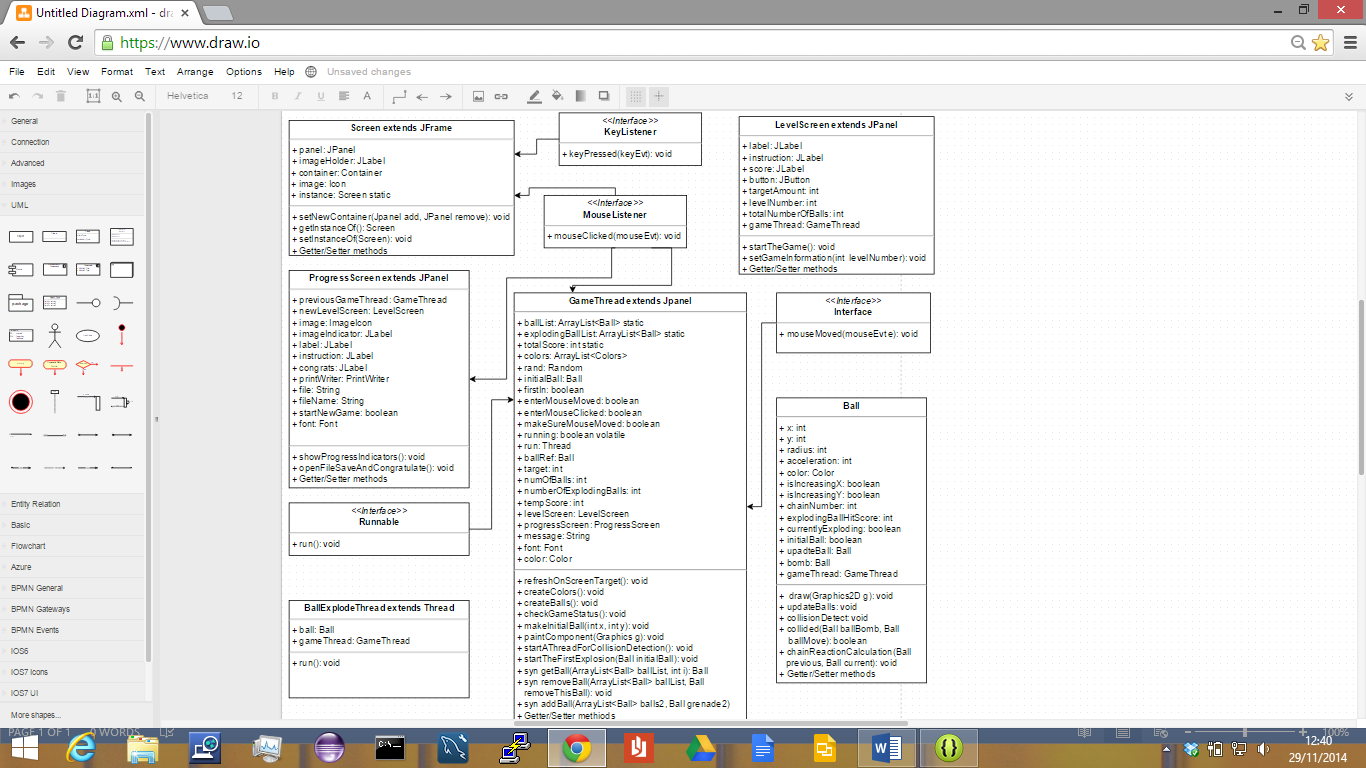


Fig.1

Fig.1 shows the class diagram for the game.

*Level Class:*

This class contains the driver and is of a singleton design pattern. This class is the frame for the game from which panels are added and removed to its content pane. This is the first thing a player sees when he launches the game. Its members are a panel to which a label is placed which in turn holds an image for the game. A static reference is also a member for accessing the methods from elsewhere. A method is defined that accepts two JPanels which adds one and removes the other to the frame. A mouse and key listener are used. A click will launch the first level screen for the game, and the key listener is used to launch a secret menu for when the user wants to skip levels. S is the secret key.

*LevelScreen Class:*

This class extends JPanel and serves to show the user what level they are at, current score and information on what is expected to pass the level. The layout is grid bag layout as this allows for more freedom when placing components. A level screen object accepts an integer in its constructor and this is used to define target amount and number of balls. Labels are used to display this information to the player and also their current score. A button is used so the user can start the game whenever they are ready. Aside from the standard get/set, two methods are defined for deciphering the game information i.e. target etc and for creating a new gameThread with this in its constructor instance and launching that thus every level screen has a gameThread object and vice versa.

*ProgressScreen Class:*

This class caters for the progress of the player throughout the game at the end of each level attempt. Mouse listener is implemented as this serves as a way for the user to press ahead with the game. A progress screen is created with a level screen, gameThread and String objects. The level screen is the level screen that should be added to the frame when the user clicks and the gameThread is the JPanel that should be removed as it is now a finished game instance. The string is used to inform the object which icon should be displayed i.e. a failure or successful image. This string is then used to open a saved image and then resize it using graphics and then added to this. If the current level passed was the last one then the passed string will be a filename and that is used to open/create a file that the current total score will be saved to as a new high score. Launching the next screen is again done by passing in this as a panel to be removed and the passed level screen object as the panel to be added. The bool start new game is used to separate between normal progression or the end of the current game in a method.

*GameThread Class:*

This class serves as the action panel where the user interacts with game and the game is active. The class extends JPanel as we want to draw on the panel and implements a runnable interface as we want to create a thread where the balls can update. Another thread is created upon the first collision which checks for collisions. These two threads run constantly until the flag is made false. In the threads the two methods are synchronized on a ball object and we use wait and notify between them. Collision detect in thread2 waits until the update method in this classes runnable has been finished and then we notify and release our lock so collision detect can be called. Repaint is also called after the updates. Mouse listeners are used to create the first explosion and the placement of the first explosion. A number of flags are used to protect the quality of the game and unintended clicks of the mouse. We use three static structures for our game. The total score needs to belong to the class as multiple gameThreads will be created but we only want one score. The ball list were also declared as static but they could easily belong to each instance also. A temporary score member is used as we need to hold our score for each level we play. Other object members are used for access and linking information for easy flow. Paint component is overridden to draw our game balls to the screen from both data structures and also any current hit score from an explosion. I have used alpha composite here to effectively mimic the transparency of the real game. Access to the ball lists need to be synchronized as well. Not only that I avoided using a iterator object to successfully alter lists while iterating through them by creating methods that accept the list and a ball/index that perform this function at an atomic level. Refresh screen target is a method that is used to give accurate information to the user during the gameThreads action. And checking the game status provides a way to decide where we go from here when the current gameThread is over.

*Ball Class:*

This class defines the ball object. It consists of an x and y coordinates, radius, colour, speed and bools to decipher what direction the ball is initially set to travel in. A ball also has a chain number member and this tells us what layer in the chain the ball has exploded and a hit score member for scoring purposes. A ball also has a constructor which accepts just a gameThread object for access purposes. A draw method is defined for drawing the ball. An update and collision detect method is defined here as all balls need to be updated and be checked for colliding into each other. A method for calculation the balls score upon collision is also defined here and that is based on a mathematical equation which uses a chain process.

*BallExplodeClass:*

This class is used for every ball that has been set to explode. The constructor accepts a ball. The gameThread object is used to provide a reference to the gameThreads methods. The thread is started and the ball is enlarged to 30 pixels radius. The thread then sleeps for 1 seconds, then it wakes up, removes the ball from the data structure and dies.