# Breakout MVP and Documentation

## Problem and Decomposition

Create an Atari Breakout like game modelling its physics, abstract design, and general gameplay logic. Design the software using Microsoft WinForms making and demonstrate understanding of the object orientated paradigm and event driven programming. The games components and features easily mould to the OOP architecture.

## Classes and Structures

|  |  |  |
| --- | --- | --- |
| Name | Derives | Description |
| **Abstract** Game | Object | Defines basic methods for an abstract game such as lists to manage objects, animations, and tasks. It’s the Games job to handle the creation and freeing of objects and components in a safe way. |
| Breakout Game | Game | An application of a game that manages the game of breakout. It creates levels, the paddle and ball, and various user interfaces. |
| **Abstract** Game Component | Object | A game component defines a component that knows what game it’s a part of a reference to random, and a Screen to draw two. Nearly all classes derive from this class to reduce repetitive parameterisation in constructors |
| Game Object | Game Component | An object with position, dimensions, velocity, and a texture that can be added and removed from a game and derived to extend its functionality. Objects can draw themselves, optionally respond to collisions, and update themselves. |
| Brick | Game Object | A derived Game objects that defines an object with density and value, that can explode with its provided texture. |
| Regrowth Brick | Brick | A brick that adds itself back into the game a set time after it has been freed. |
| Main Menu | Game Component | A component that manages a collection of buttons, toggles, and text to allow the user to choose what to do when the game is run. Such as play, view guides, configure the game, or view the credits. |
| Cursor | Game Object | A Game object that follows the mouses position so the user can see where the real cursor is (as it is hidden) |
| Text | Game Object | A Game object that renders a collection of game objects with lettering textures to display a message on the screen. Managing the creation, updating, and removal of such objects. |
| Button | Text | A text objects that calls the provided action delegate (call-back function) when the mouse clicks within its collision box (AABB) |
| Toggle | Button | A Button with state. Calling the provided action delegates when toggled and animating the toggle texture. |
| Paddle | Game Object | A Game object that follows the mouses Y position and takes part in the games physics simulation. |
| Ball | Game Object | A Game Object that bounces off bricks, walls, paddles, and worms. Applying damage to bricks and decrementing lives when traveling below the paddle. |
| Worm | Game Object | A Game objects that moves across the screen horizontally with random wait times participating in the physics simulation. They appear in the third level. |
| Backdrop Manager | Game Object | scrolls a game object texture over the screen in a conveyor type cycle to give the illusion of an infinite backdrop. |
| Animation | Game Component | Manages the switching of a collection of textures on a provided game object. Can loop infinitely, a set number of times, and optionally call actions when done animating. |
| Level | Game Component | Groups a collection of bricks together and their arrangement on the screen and decides when bricks should drop augments. |
| Second Level | Level | Extends the functionality of a level, adding regrowth bricks along the bottom row of regular bricks. |
| Third Level | Second Level | Extends the functionality of the second level by spawning 3 worms below the regrowth bricks for extra difficulty |
| Screen | Object | Bridges the drawing of objects with the forms graphics objects. Managing scaling and the position of the mouse and click events. |
| Form 1 | Form | Creates the Breakout Game, starts the timer, and calls the Main game loop per timer tick. The form passes information such as click events and mouse positions into the Breakout games screen. |
| **Structure** Vector 2D | - | A simple structure to manage a vector in 2d space with zeroing and inverting methods. |
| Task | *Object* | Stores an action delegate to be called by the Game class a set number of milliseconds later. A simple form of asynchronous programming without threading. |
| **Enum** Direction | *-* | Stores a group of constants that act as cardinal directions. |
| Tile Set | *Object* | Manages the provided image as a group of tiles, returning texture source rectangles to game objects and animations. |
| **Static** Time | - | A static class to group a collection of constants referencing time in milliseconds |
| Augment | Game Object | A game object dropped from bricks by a level. Augments plug in new code when collided with by the paddle and reject such code when a condition is met. |
| Triple Ball Augment | Augment | Adds 2 extra balls to the game, animates the balls and paddle, and rejects when there are less than 2 balls left above the paddle (also when the game ends or level changes) |
| Exploding Ball Augment | Augment | Causes random bricks to be deleted from the current level when the ball collides with any brick. Rejecting after its application. The ball and paddle animate while applied. |

## Functionality

The player will gain a point each time the ball collides with a brick, the value increasing for denser bricks (more hits to destroy). Bricks will award 12 points for the first hit, with subsequent hits awarding 12 mor than the previous. Some bricks will destroy after a single hit, with others after two and three.

Although the brief suggested 10 points per brick hit, I decided to use 12 as it is my favourite number.

The player wins a level when they have destroyed all the generated bricks, then proceeding to the next level. The player wins the game when they have beat all the provided levels (there will be three in this case).

The first level has minimal functionality, with 6 rows of bricks. The second level introduces regrowth bricks that ‘regrow’ after they’ve been destroyed to be hit again. They award no points and do not deflect the ball when collided with positive vertical velocity (downward ball). The third level expands the second levels functionality by adding annoying worms that traverse the screen horizontally below the rows of bricks. They merely deflect the ball on Collison and are not destructible for points.

If the user specifies the ‘single level mode’ option. Only the first level needs to be completed for the game to be won.

The player can lose a life if the ball falls below the screen height, losing the entire game if running completely out of lives (three in this case). If the Triple ball augment is active, the player will only loose a life when the last ball goes off the screen.

### Form Design

Although built in windows forms. Main use of the provided components will be phased out used to ensure the games aesthetic matches that of a pixel perfect recreation of breakout. This will involve the creation of custom classes such as *Buttons* and *Text Boxes* so it will not be recognisable as a windows form’s app in any way.

The game will make use of a single *Bitmap* to render the game content using *Graphics* objects with double buffering with a single *Timer* to handle the main game loop. I plan to push all the game code out of the *Form1.cs* file and use a separate *Screen* class to manage rendering to completely detach the *Game* class from *Graphics* and *Image* objects.

## Minimum Viable Product

Like mentioned in the **Classes and Structures**section the MVP follows a OOP architecture of 5 classes with the following UML diagram and form design

A picture containing graphical user interface

Description automatically generatedA screenshot of a computer

Description automatically generated

The MVP doesn't have a score, lives, or ability to win or lose. I just created the physics and destruction of bricks. The MVP has main visual functionality of Breakout without terribly extravagant features.

## Final Product Class Diagrams

Found in Breakout Project—*ClassDiagram1.cd*