LEP and the Bugs

IT 5107 Programming Essentials for Artificial Intelligence

G.W.S.S. Keshan De Silva – MSCAI/14/15

LEP AND THE BUGS IS A SIMPLE BUBBLE SHOOTING GAMES. PLAYER'S GOAL IS TO FIRE THE BUBBLE IN TO THE BUGS BEFORE THEY PASS THROUGH

Table of Contents

Introduction

LEP and the Bugs is a simple bubble shooting games. Player's goal is to fire the bubble in to the bugs before they pass through. LEP represented by a gun, player can move the gun up and down as well as can rotate limited angles. With different amount of power gun can release the bubble in a projectile manner.

RULES AND STATES

LEP and the Bugs Game contain few simple rules. LEP or the Gun

1. Can only move up and down on given line
2. Can rotate from 0^o to 90^o
3. Force can be adjusted for a given range

Player can select any combination of above configurations and fire the bubble. If the bubble directly hit on a bug that bug will be defeated and marks will be added to the player.

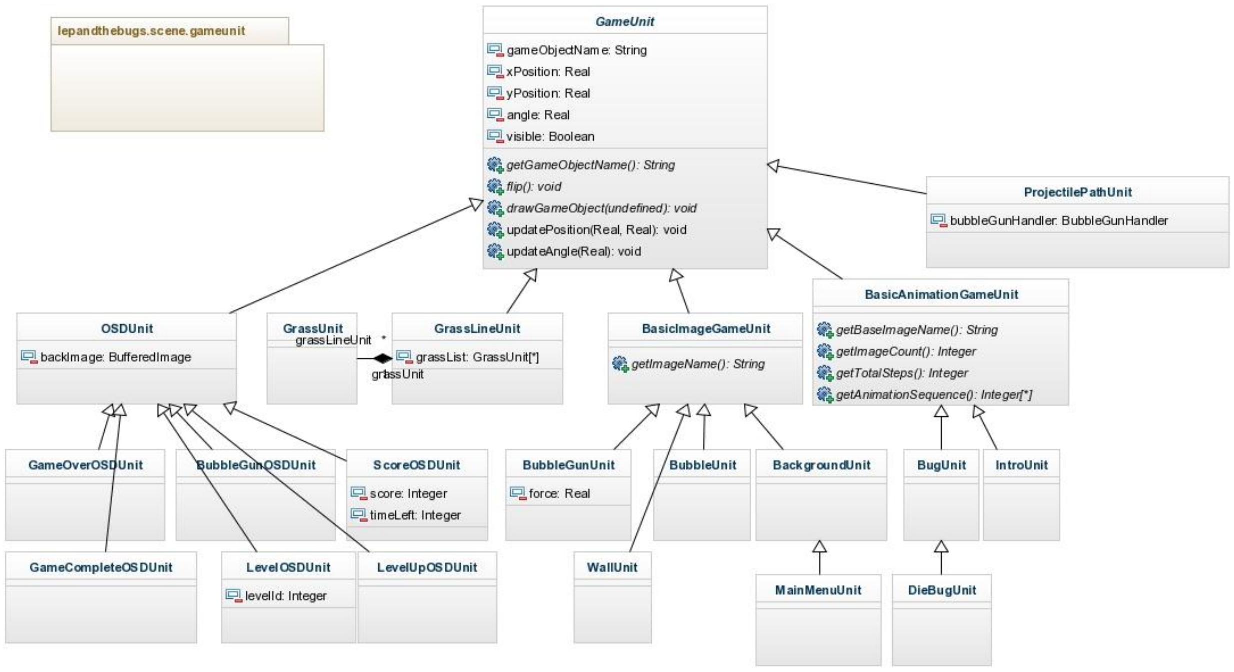
If player manage to defeated all the bugs within the time duration, then that level will be completed and move to the next level of the game. Completing all the levels in a row will make player win the game. If player fail at least one level play must restart from the beginning (First Level).

Following conditions will cause a failure of a level.

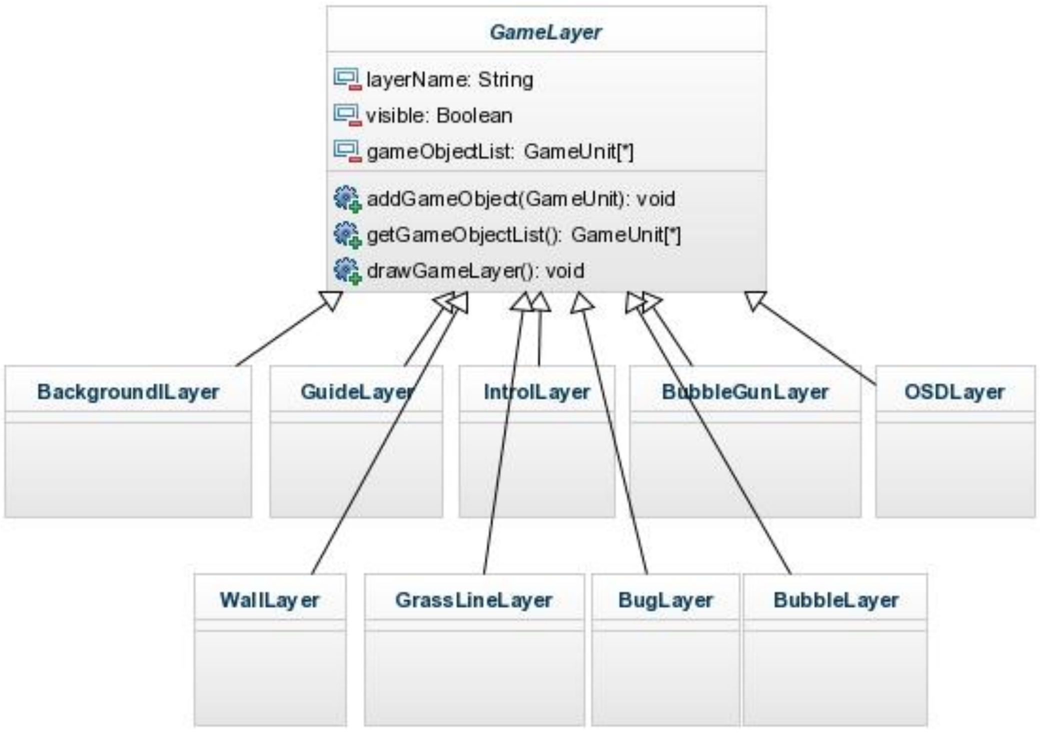
1. Player running out of time.
2. At least one bug manages to escape from the scene.
3. Player uses all 12 bubbles and even after that there are more live bugs.

DESIGN

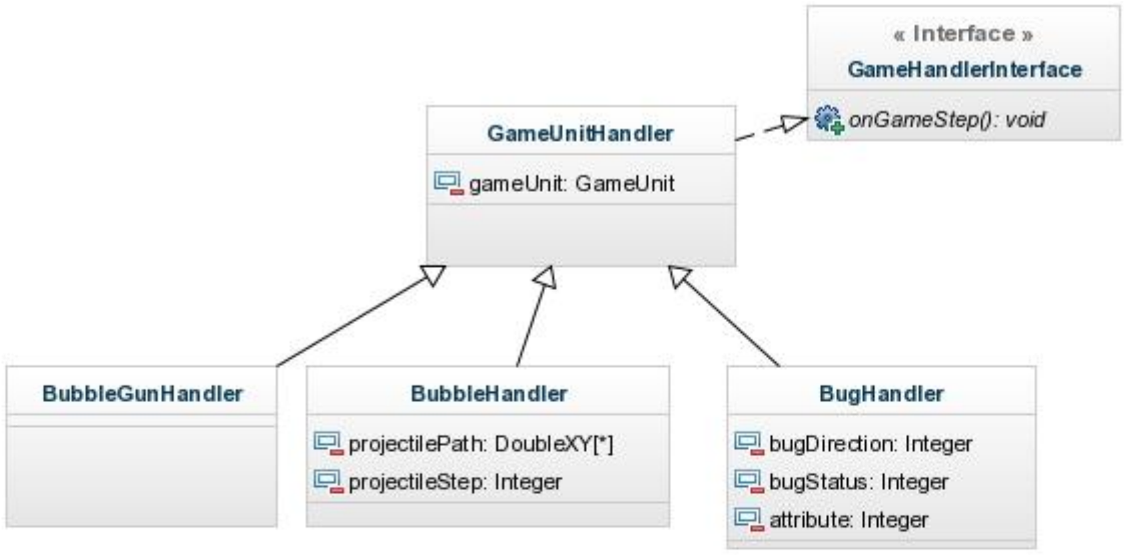
Design ***of the game, LEP and the Bugs*** contain several classes and interfaces. All the Graphical representation of the Game UI is describes under the Game Unit class diagrams. Abstract class 'Game Unit' will provide the basic functionalities for all units.



Game Unit will be added to their respective Game Layer, in order to draw the game units on the game scene. Each Game Layer is a subclass of abstract class 'GameLayer'. This calss will provide all the basic functionalities to the layers.

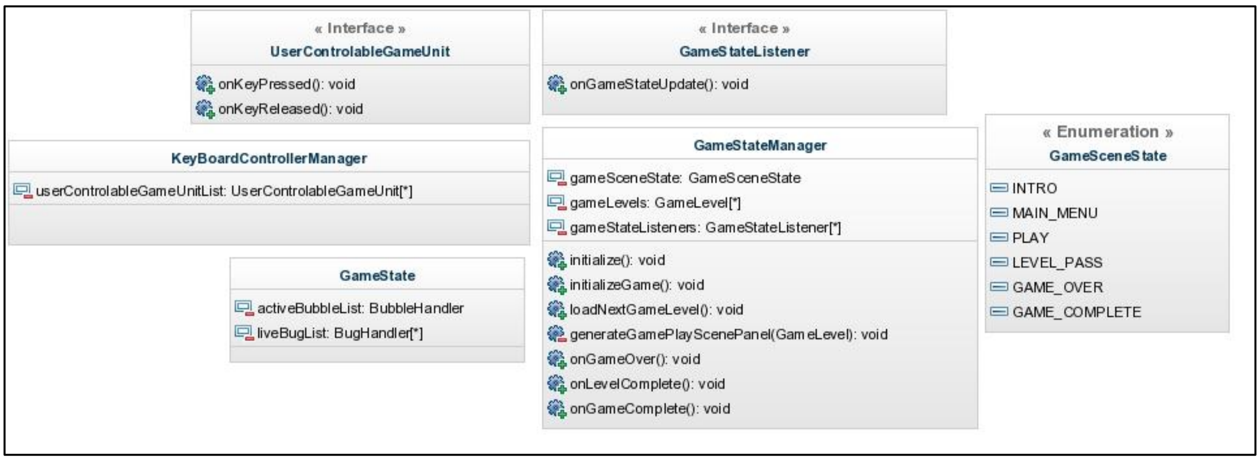


Each and every Game Unit has its own Game Handler. Game Handler will communicate with the game engine and update necessary properties of the game unit.

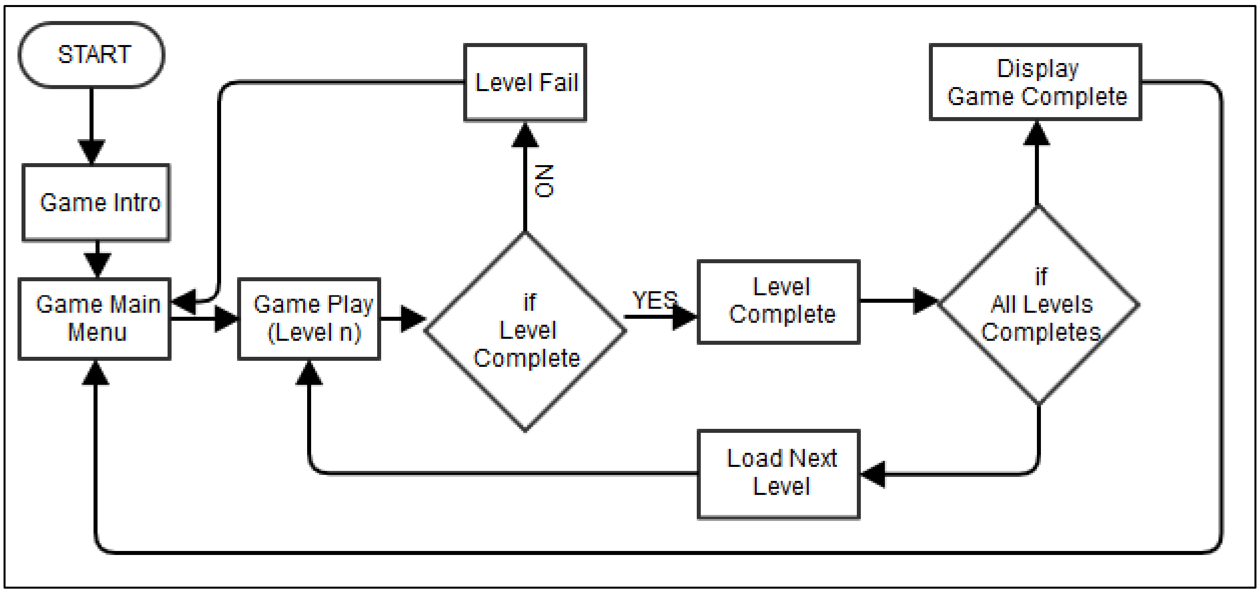


Since Game Layer and Game Scene use those properties to generate the Game Unit on the screen, Game Handler can change the Game Unit appearance on the game without directly interact with the Game Scene. This concept helps to completely separate the Game UI is from the Game Engine.

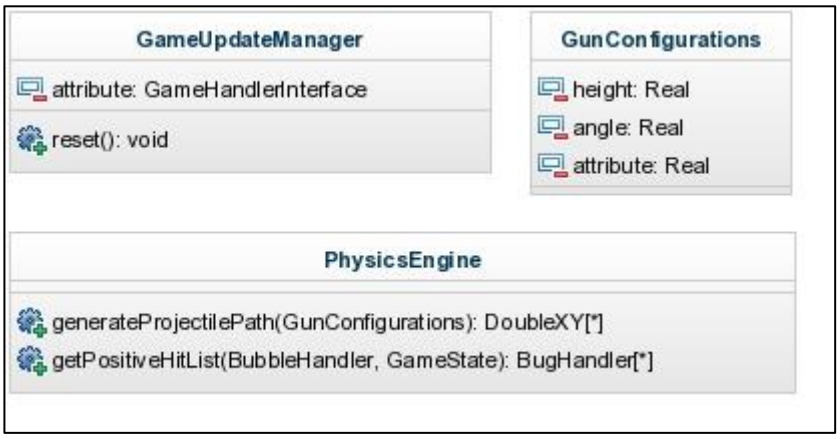
There are set of Manager Classes which are responsible for managing the game.



Keyboard Controller Manager Class will handle all the key board input from user and redirect those actions to necessary game units / Game Handlers. Game State Manager will handle all state changes of the game. Simple flow diagram for the design is given below.



Engine classes take the responsibly of calculating physics and update Game dynamics. Basically generating the projectile path for to given Gun Configuration and determine the positive hits vs. Negative hits on each projectile.



GameUpdateManager

attribute: GameHandlerInterface

reset(): void

GunConfigurations

height: Real
angle: Real
attribute: Real

PhysicsEngine

generateProjectilePath(GunConfigurations): DoubleXY[*]
getPositiveHitList(BubbleHandler, GameState): BugHandler[*]

Game Update Manager is responsible to call onGameStep on each Game Handlers. This method call will cause game handlers to update the Game Unit properties if necessary.

IMPLEMENTATION

Implementation of the LEP and the Bugs Game was done by the java programming language in Neatbeans IDE. All the classes discuss in the design part was implemented with necessary attributes which are additions to the above class diagrams.

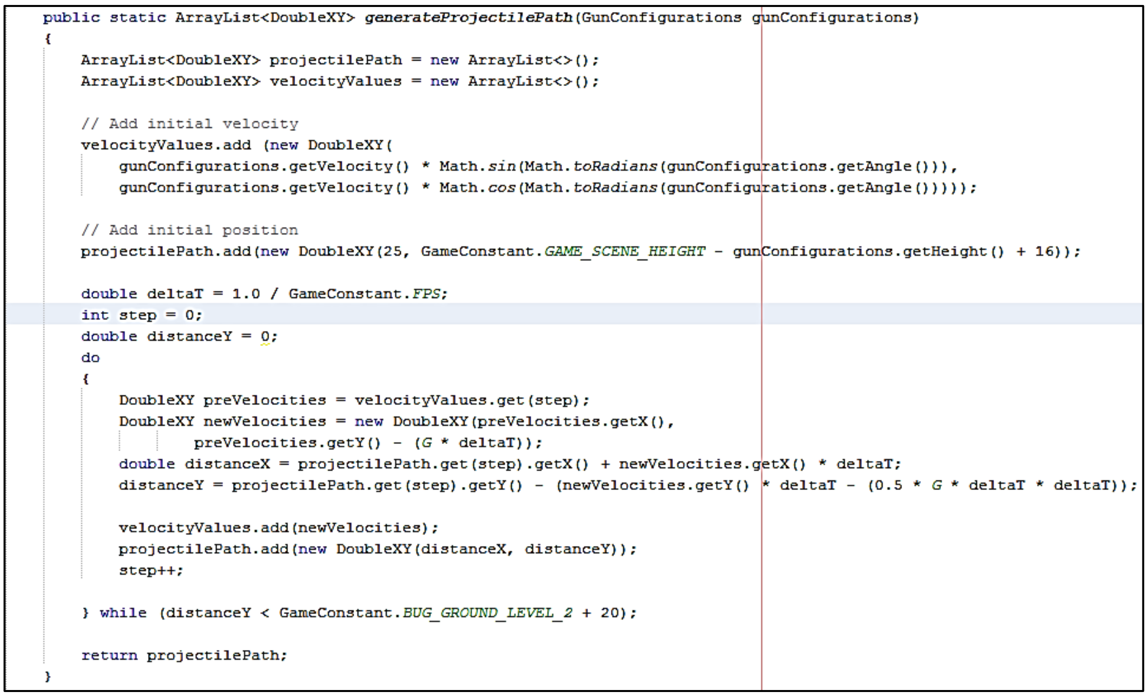
GAME SCENE IMPLEMENTATION

Primitive objects in the game will be represented by the GameUnit class. Where it has `public void drawGameObject(Graphics graphics)` method. That method will draw the game unit in to the graphics.

GameLayer which contains set of game units, will be added to the scene. Scene is a basic Java JPanel, because of that it has its own `public void paintComponent(Graphics graphics)` method. This method will be called whenever refresh method is invoked either by program itself or the player action. paintComponent method will call all the layers and retrieve all game units from those layers. After that all the Game Units will be drawn on the Game Scene.

GAME ENGINE IMPLEMENTATION

`Game` engine contains methods to calculate the projectile path for given gun configuration. In addition to that separate method will be added to determine whether hit is positive or negative.



```java
public static ArrayList<DoubleXY> generateProjectilePath(GunConfigurations gunConfigurations)
{
    ArrayList<DoubleXY> projectilePath = new ArrayList<>();
    ArrayList<DoubleXY> velocityValues = new ArrayList<>();

    // Add initial velocity
    velocityValues.add (new DoubleXY(
        gunConfigurations.getVelocity() * Math.sin(Math.toRadians(gunConfigurations.getAngle())),
        gunConfigurations.getVelocity() * Math.cos(Math.toRadians(gunConfigurations.getAngle()))));

    // Add initial position
    projectilePath.add(new DoubleXY(25, GameConstant.GAME_SCENE_HEIGHT - gunConfigurations.getHeight() + 16));

    double deltaT = 1.0 / GameConstant.FPS;
    int step = 0;
    double distanceY = 0;
    do
    {
        DoubleXY preVelocities = velocityValues.get(step);
        DoubleXY newVelocities = new DoubleXY(preVelocities.getX(),
                preVelocities.getY() - (G * deltaT));
        double distanceX = projectilePath.get(step).getX() + newVelocities.getX() * deltaT;
        distanceY = projectilePath.get(step).getY() - (newVelocities.getY() * deltaT - (0.5 * G * deltaT * deltaT));

        velocityValues.add(newVelocities);
        projectilePath.add(new DoubleXY(distanceX, distanceY));
        step++;

    } while (distanceY < GameConstant.BUG_GROUND_LEVEL_2 + 20);

    return projectilePath;
}
```

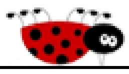
SEPARATION BETWEEN GAME ENGINE AND THE GAME SCENE

GameUpdateManager which is having a timer fires at interval 1000/32 seconds. When that timer hits it will call onGameStep() on each and every Game Handlers. Inside onGameStep() they will be updating their properties such as PositionX or Angle.

On the other side FrameManager will call refresh method in all registered GameUnit with the same rate (1000 / 32 seconds). When it redraw the game unit it will reflect all the property changes has been done to the GameHandler throughout the Game.

Game Units

Game contains several game units which may be static or dynamic. In addition to that some of the units can be controlled by the user. Following table contains list of Game units with simple description.

Game Unit	Description
	Bubble gun which is controlle by the user. User can move UP DOWN or rotate the gun. In addition to that user can increase or decrease the initial velocity of the fire.
	Bubble : This will be fired from the Bubble Gun
	LIVE Bug : Live bugs (can be static or dynamic-moving) will display lower of the game scene. Once they hit with a bubble they converted to a DIE Bug
	DIE Bug : Once bug hit with a bubble they will converted to a DIE Bug

Sample UI from LEP and the Bugs

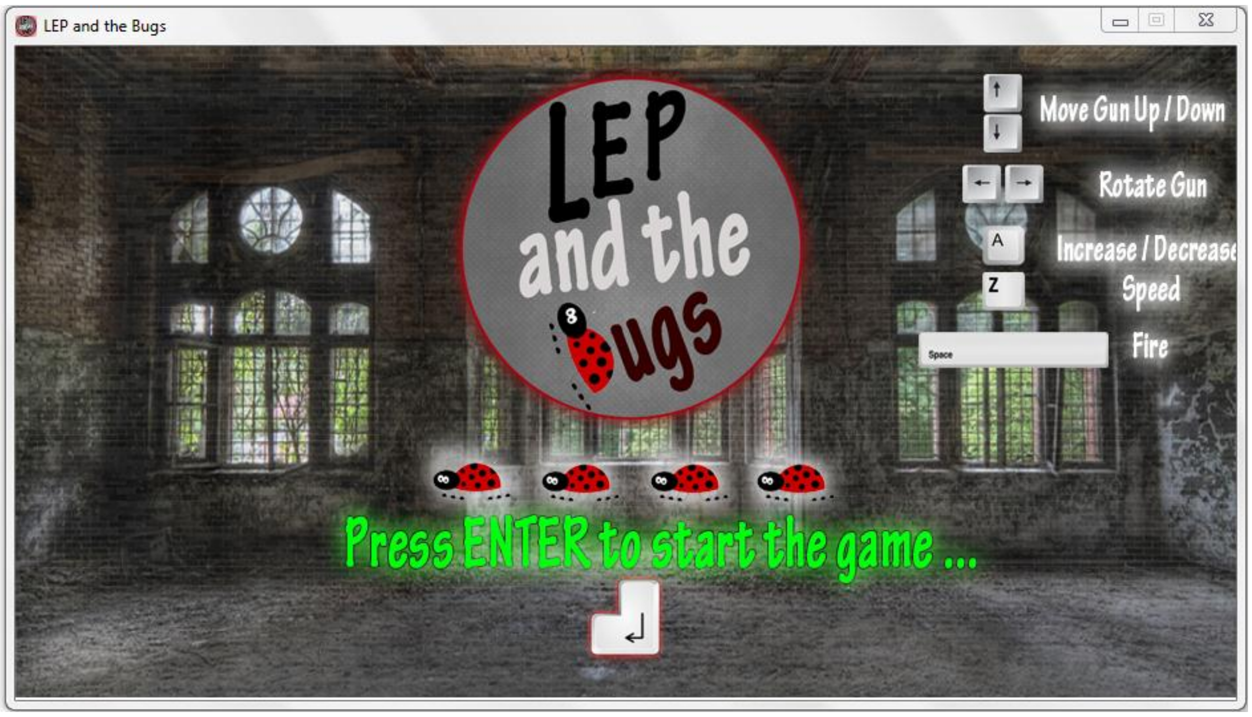


Figure 1 : Main Menu of the Game

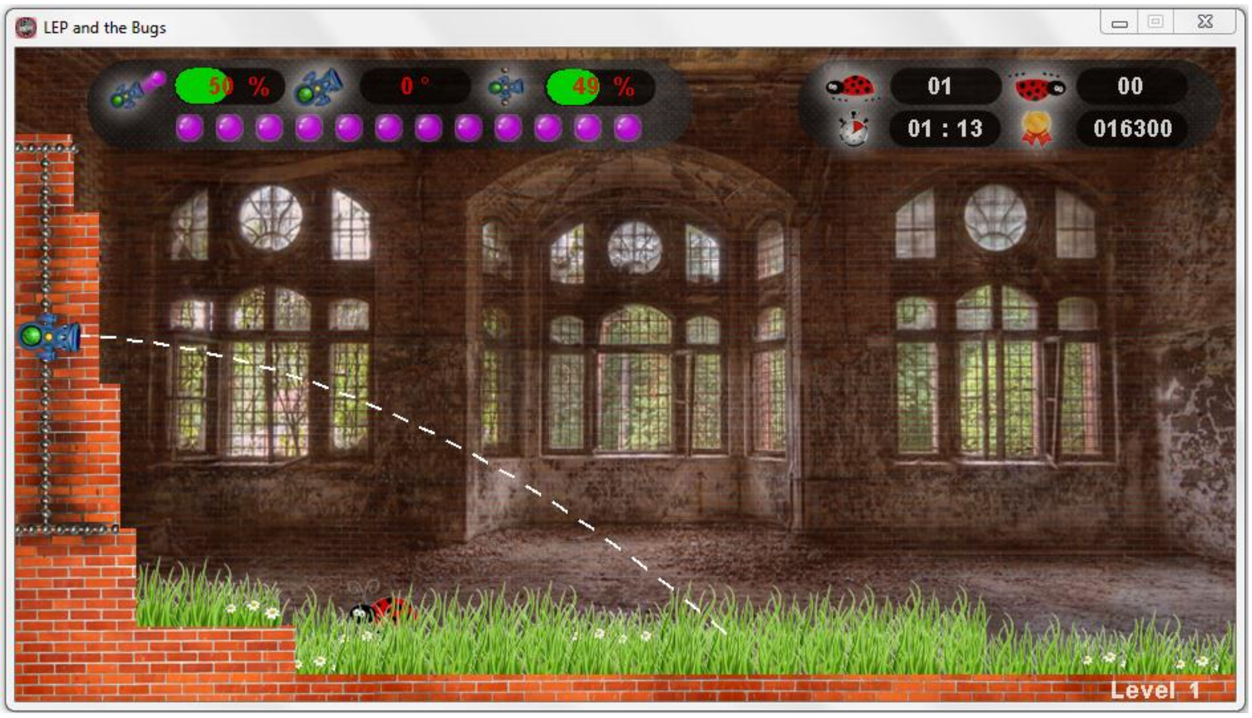


Figure 2 : Level 01 Playing

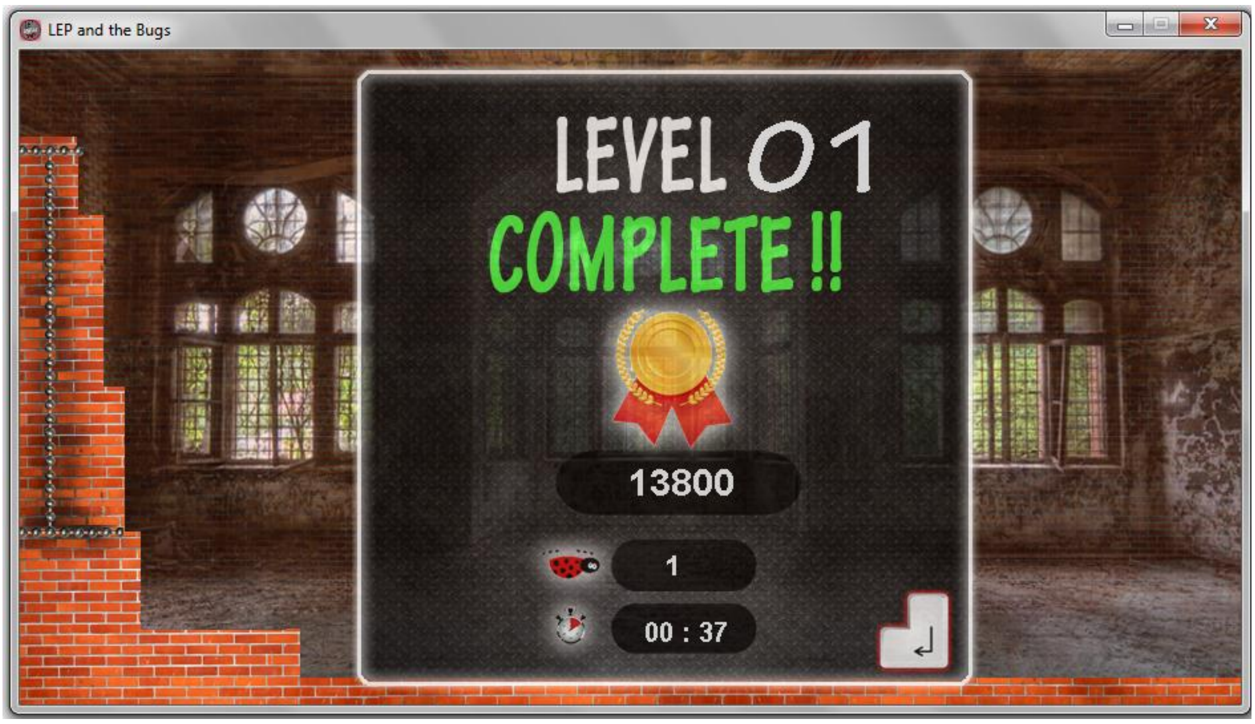


Figure 3 : Level 01 Completed

FUTURE WORKS

- Implement a score storing system where player can add there highest scores
- Add a simple music / sound-fx components to the game.
- Introduce different types of Bugs with the level increment.

GAME USER GUIDE

BASIC KEY'S

Key	**Operation**
↑ ← ↓ →	User can control Gun Up/Down movement using UP key and Down Key. Rotation of the Gun Can be controlled buy LEFT and RIGHT keys
A Z	User can Increase and Decrease the Initial Velocity of the bubble by using A and Z keys.
Space	User can fire a bubble by using space bar

SCORE

Initial score will be 3000, in addition to that

- 100 points will be added for each remaining seconds.
- 500 points will be offered for each remains bubble.
- 2000 points will be offered for each positive hit.

LEVEL

LEP and the Bugs having total of 10 levels, user can progress through the levels by completing one after another.

Level features will be differed based on the number of bugs and the static / dynamic behavior of those bugs.

In addition to that one Game world will only contains two levels. So that after two levels game world background will be changed.