# SCIENCE FOR GAMING

A final project submitted to the University of Wales in partial fulfilment of the requirements of BA (Hons) Digital Media in Game Development

**June 2013**

## Blosics

**Pavankumar Chopra**

**2nd Year UG – Game Development**

**11UG03040**

**ICAT, Design and Media College,**

**Bangalore**

# Table of Contents

BLOSICS ..... 3
OVERVIEW ..... 3
MARKETING ..... 3
GAME PLAY ..... 3
GAME RULES ..... 4
GAME CONTROLS: ..... 4
GAME TITLE ..... 4
TECHNICAL SPECIFICATIONS ..... 4
FUTURE ENHANCEMENT ..... 4
GAME ASSETS ..... 5
SCREENSHOTS ..... 6
LEVEL SCREENSHOTS ..... 7
UML DIAGRAMS ..... 11
USE CASE DIAGRAM ..... 11
ACTIVITY DIAGRAM ..... 12
STATE DIAGRAM ..... 13
CLASS DIAGRAM ..... 14
SEQUENCE DIAGRAM ..... 15
SOURCE CODE ..... 16

# BLOSICS

## OVERVIEW

Blosics is a casual physics based object-flinging game like Angry Birds. Shoot at structures built from colored blocks and make them drop off the platform. It sounds simple at first but beware - the blocks are alive! Use appropriate force to get rid of them!

## MARKETING

**Target audience**

**Age**: 5-plus

**Gender**: Both

**Geographic target**: World-Wide

**Platform-** Windows OS

## GAME PLAY

Each level in the game has a “Ball Summoning Area”. The ball can be launched only from this area, making the game challenging. The player must click, drag and release to launch the ball towards the structures. The click must be registered within the summoning area. The structures are built out of blocks of varies shapes and colours. These blocks are placed on one or more platforms. Player must earn 100 points to advance to next level. Player is awarded 10 points for each block falling off the platform, whereas each ball launched costs 12 points. So, the player must collect 100 points by using the least number of balls possible as, each ball drains the points by 12.

In case, player feels that he has wasted a lot of balls and can’t collect 100 points, he can restart the level by clicking on the Restart button at any point of time. On collecting 100 or more points, the player can advance to next level by clicking on Next Button. This button is not active, if points are less than 100.

The location of the game scene is a Rain forest and the weather is rainy, which is implemented using a Particle System.

**NUMBER OF LEVELS:** 7

## GAME RULES

1. The ball can be launched from BALL SUMMONING AREA only.
2. Player must collect 100 points to advance.
3. Player is awarded 10 points for each block falling off the platform, whereas each ball launched costs 12 points.

## GAME CONTROLS

**Click** – Spawn the Ball. **Drag** – Apply force. **Release** – Launch the ball

## GAME TITLE

Blosics

## TECHNICAL SPECIFICATIONS

• OS: Windows 7/XP PC

• Processor: 1GHz Intel Atom or AMD CPU

• Graphics Card: DirectX 7 or above compatible Nvidia or AMD ATI card

• Graphics Card Memory: 32MB

• RAM: 256MB

• Sound card: DirectX compatible sound card

• 100 MB free HDD space

## FUTURE ENHANCEMENT

Snow Particle System will be implemented in the next version of the game. Few more levels will also be added.

# GAME ASSETS

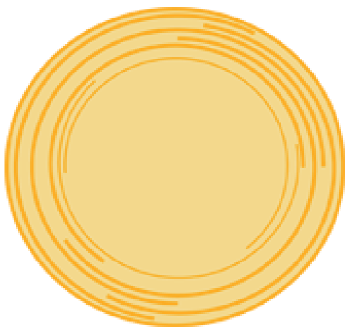
1. Ball Spritesheet

2. Square block Spritesheet

3. Rectangular block Spritesheet (Vertical)

4. Rectangular block Spritesheet (Horizontal)

5. Launch Pad

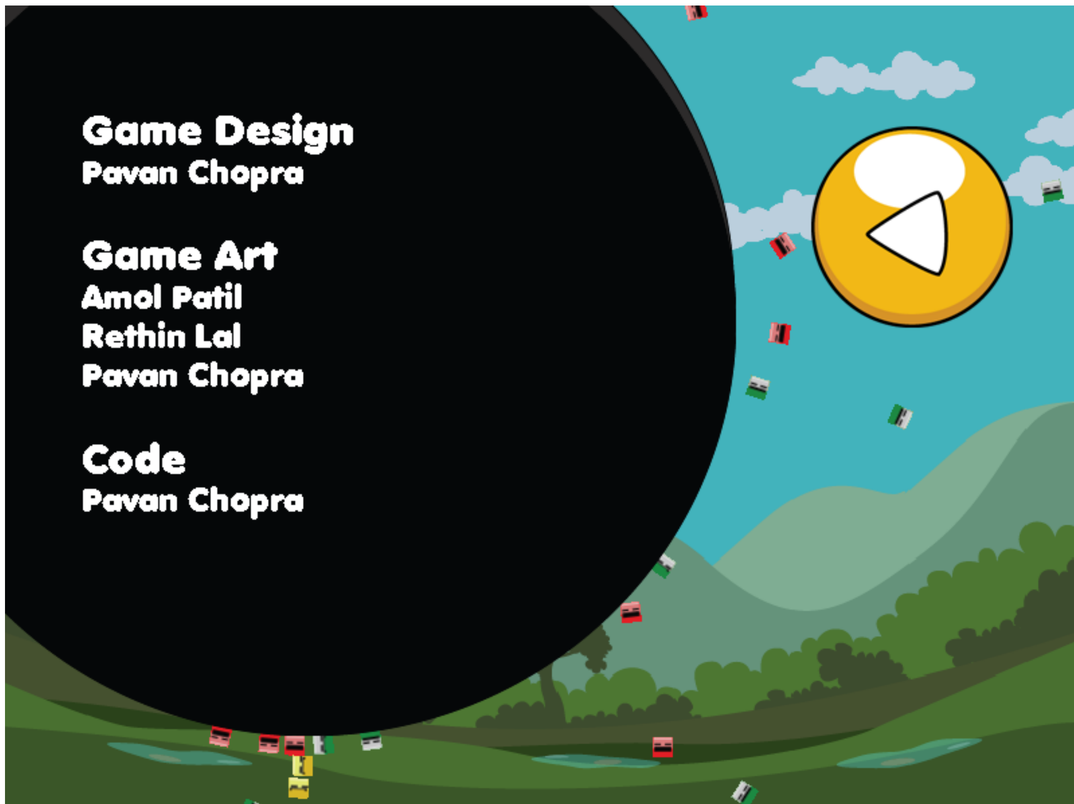


6. Platform

## SCREENSHOTS

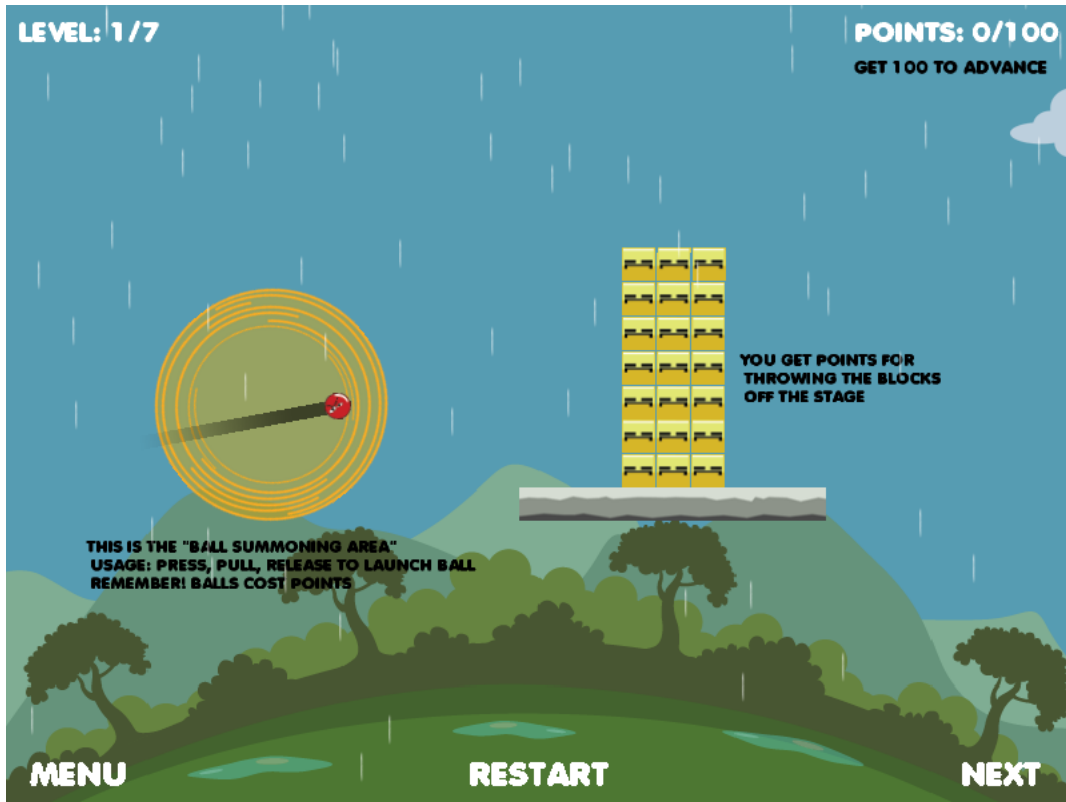
**MAIN MENU**

**CREDITS**

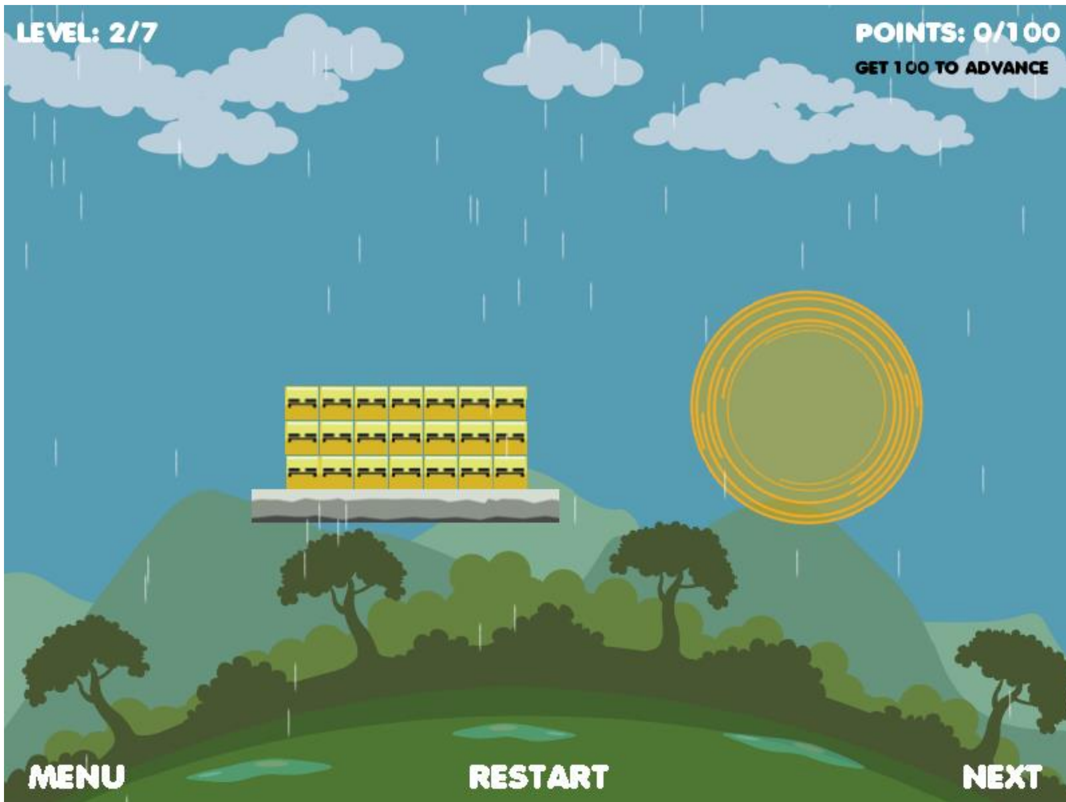


## LEVEL SCREENSHOTS

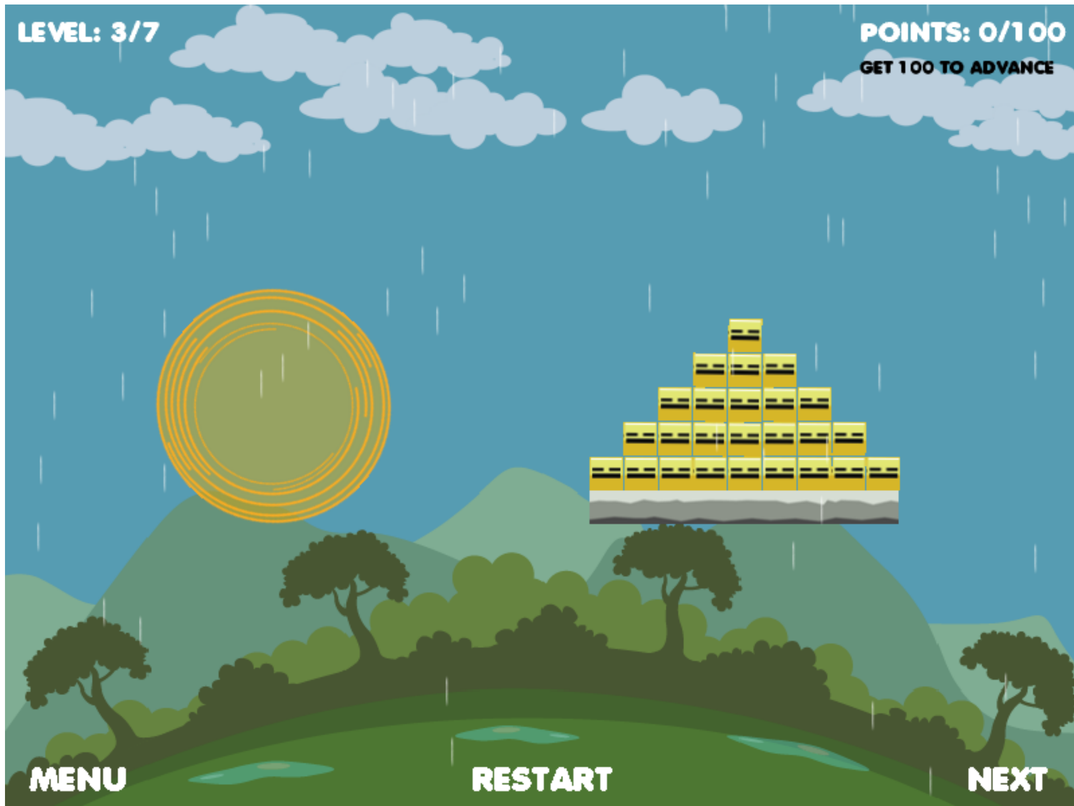
**Level 1**



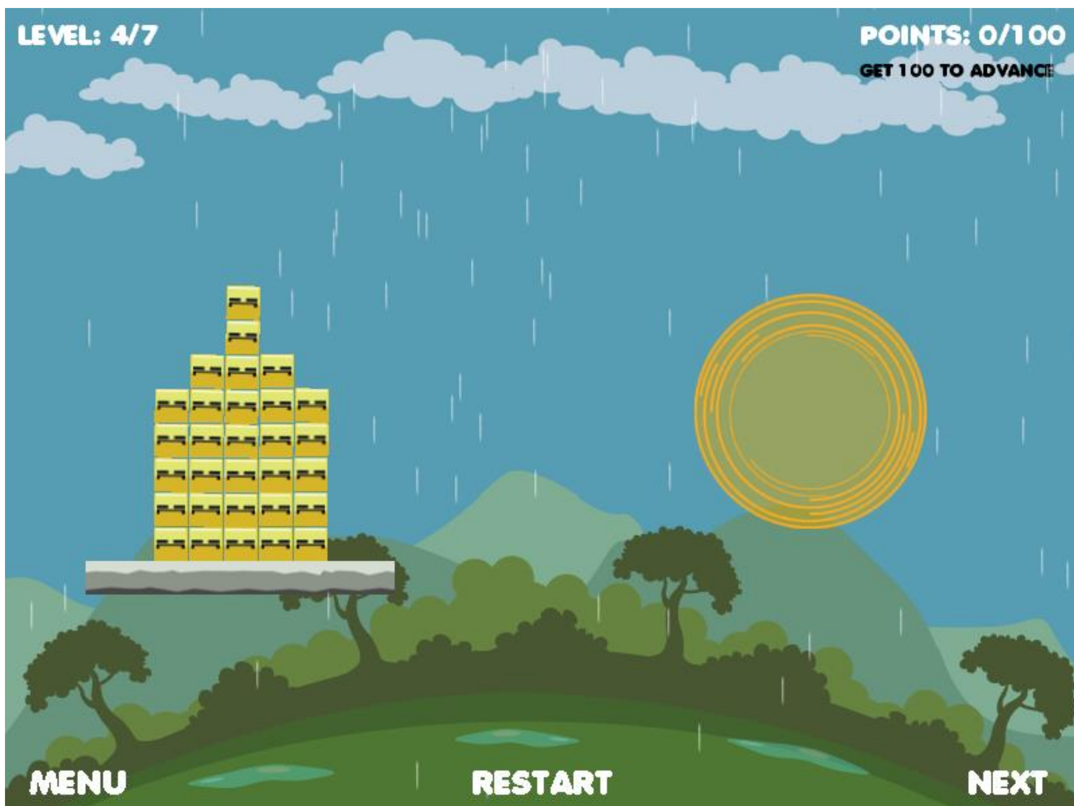

**Level 2**



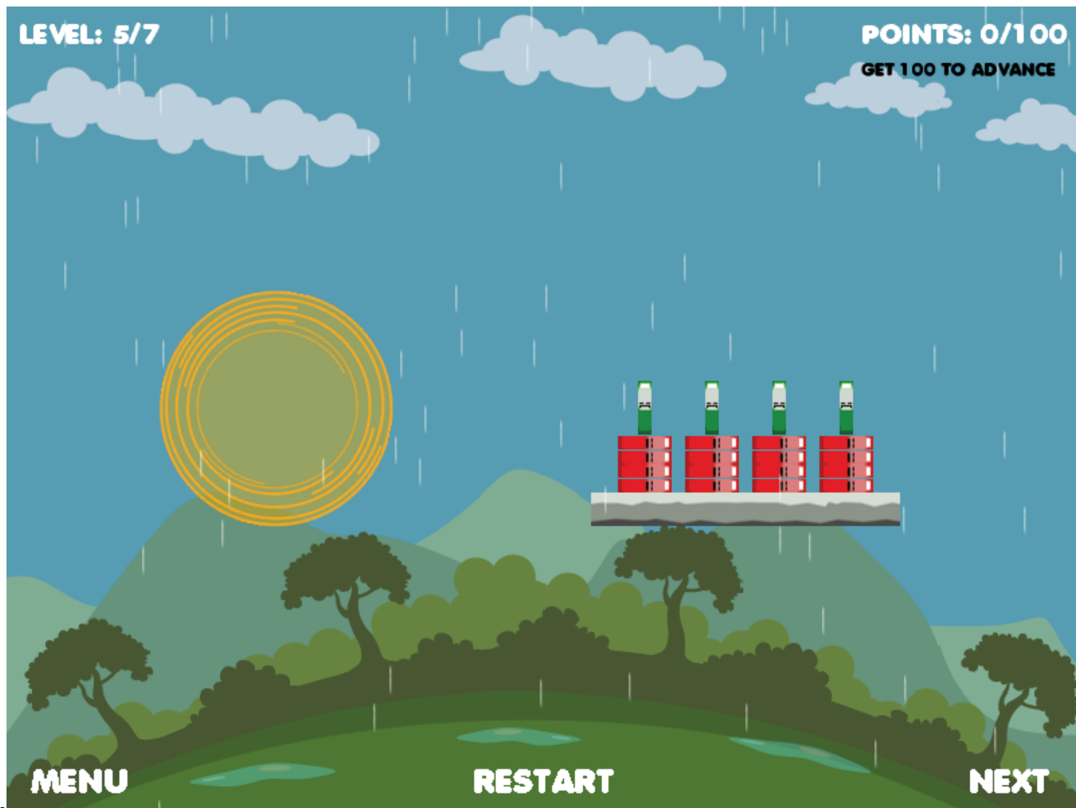
**Level 3**



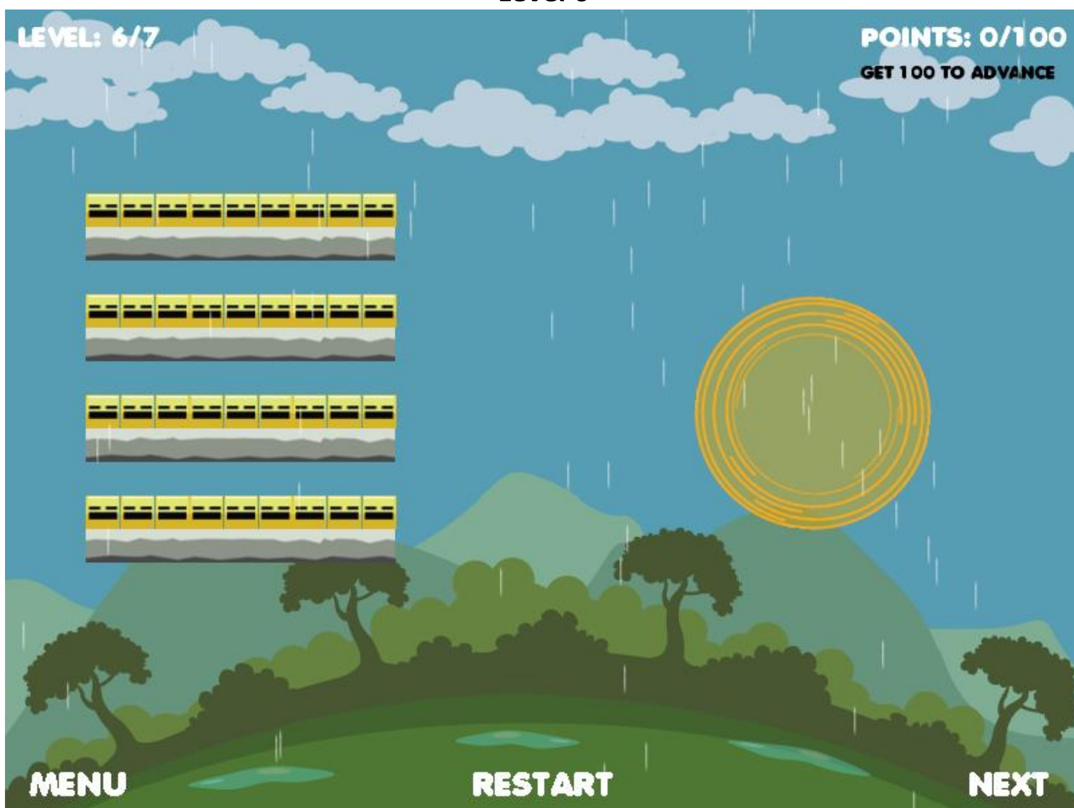

**Level 4**



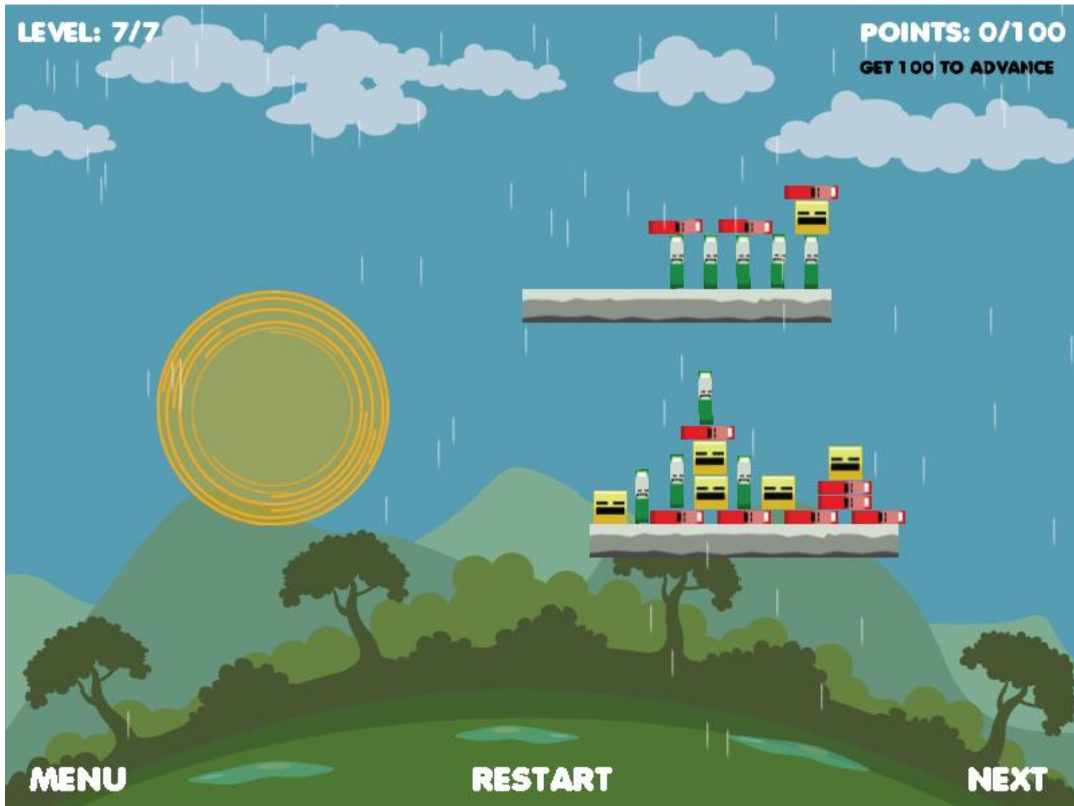
**Level 5**




**Level 6**



**Level 7**

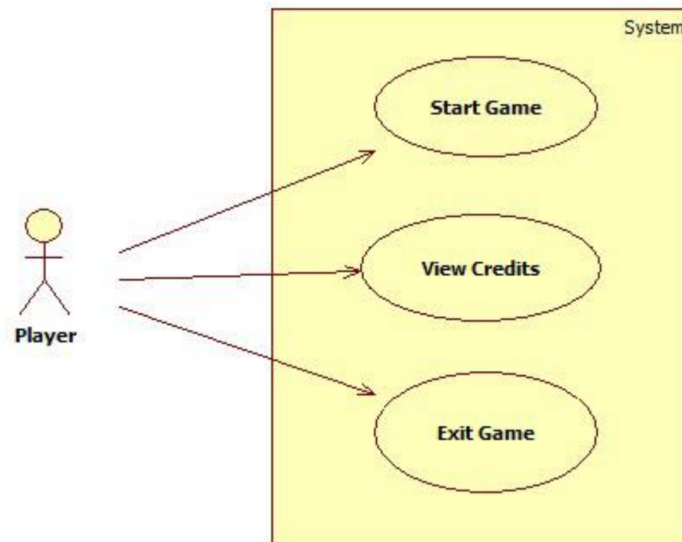


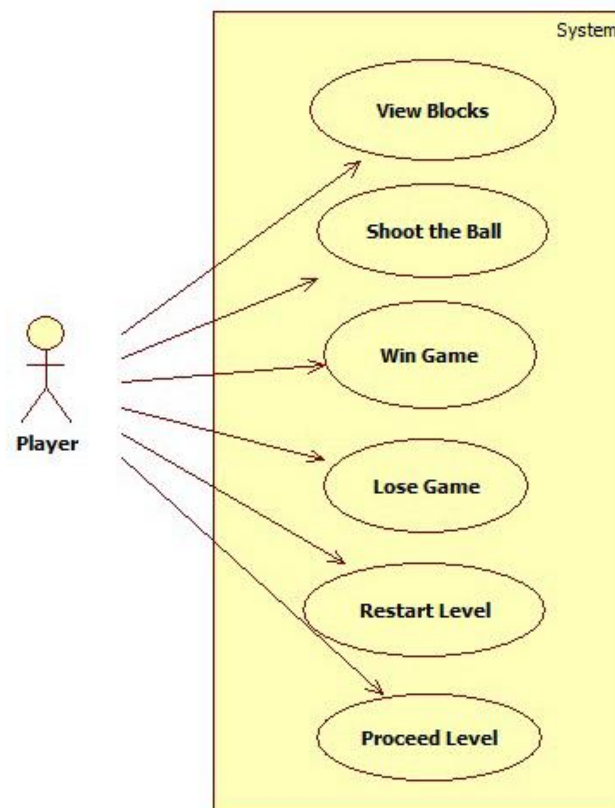
# UML DIAGRAMS

## USE CASE DIAGRAM

**Description:** The below diagram represents the actions, user can performed in the game. This diagram helps us to know all the actions user can perform in the game.

**Diagram:**

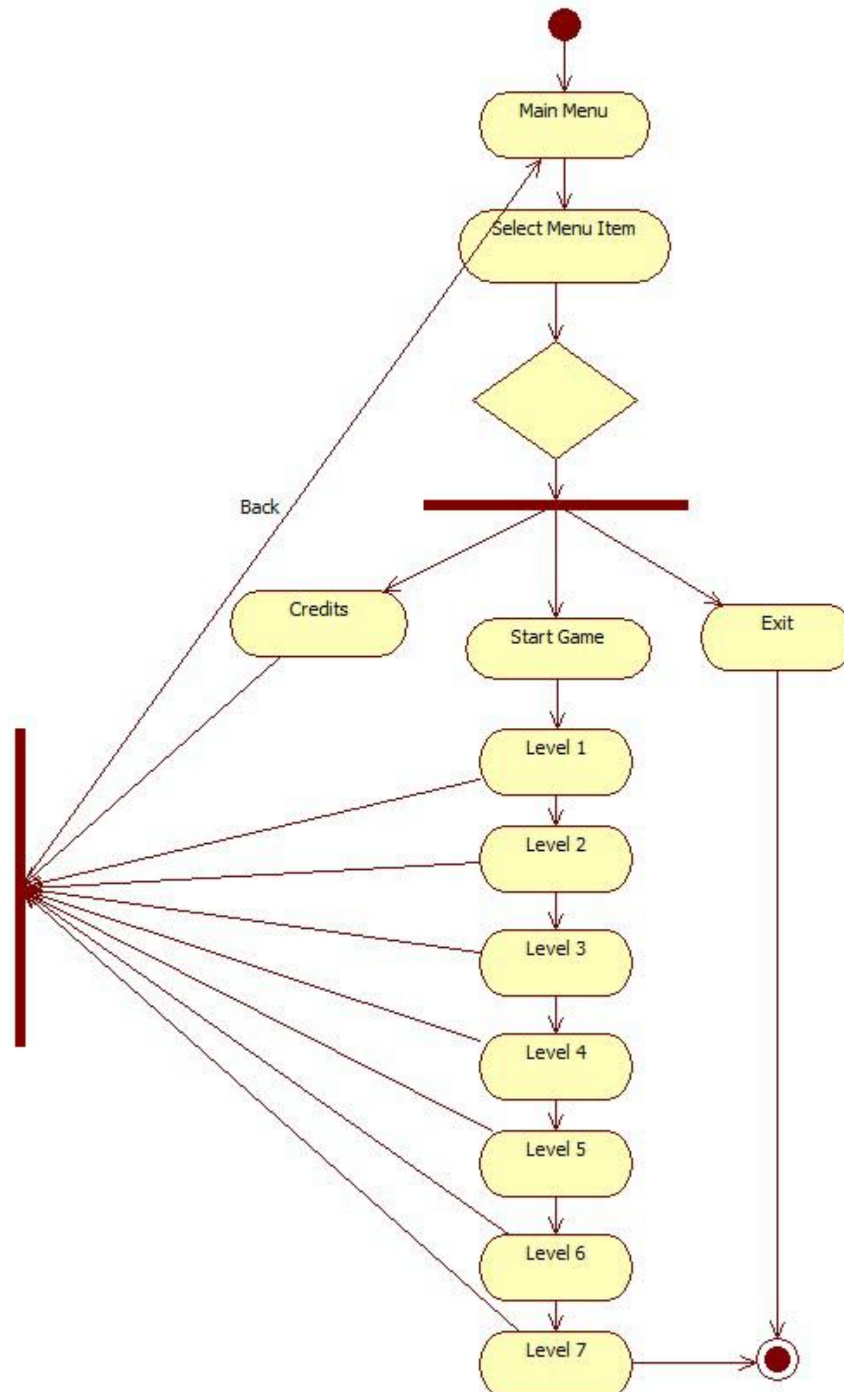


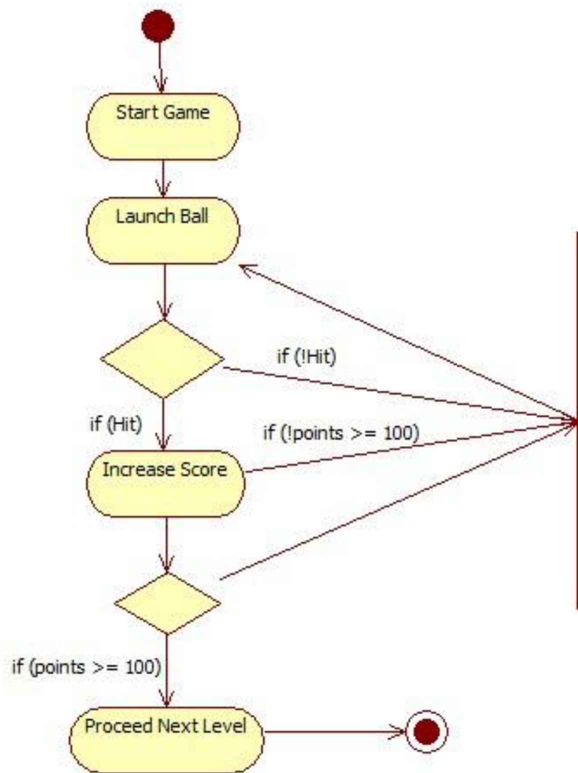



## ACTIVITY DIAGRAM

**Description:** Activity Diagram is used to show the game-play mechanics working parallel.

**Diagram:**

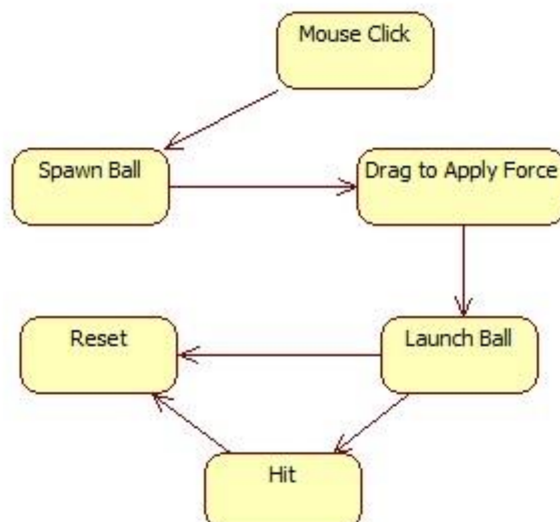





## STATE DIAGRAM

**Description:** The diagram represents States of the Player. Activity Diagram helps us to know what activities can be performed by a Player or an AI.

**Diagram:**



## CLASS DIAGRAM

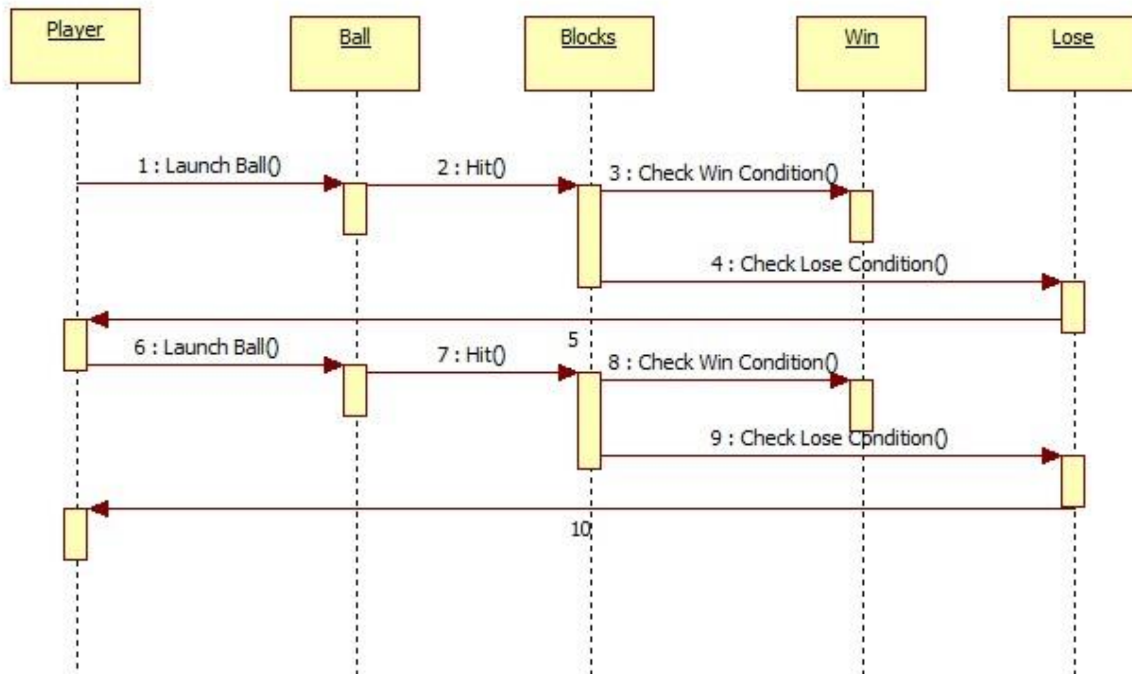
**Description:** The below diagram represents the class diagram. Then some important variables in the class and then functions declared in the class.

**Diagram:**

## SEQUENCE DIAGRAM

**Description:** Sequence Diagram helps user to know the interaction between Player, Ball and Blocks.

**Diagram:**



# SOURCE CODE

## Button.cs

**//Handles Button States**

```
        public enum buttonState
        {
            HOVER,
            UP,
            DOWN
        }
```

**//Handles Rectangular Button Touch Events**

```
        public bool Touched(MouseState touch)
        {
            currentState = buttonState.UP;
            texture = texture_up;
            if (rectangle.Contains((int)touch.X, (int)touch.Y))
            {
                currentState = buttonState.HOVER;
                texture = texture_hover;
                if (touch.LeftButton == ButtonState.Pressed)
                {
                    currentState = buttonState.DOWN;
                }
                return true;
            }
            return false;
        }
```

**//Handles Circular Button Touch Events**

```
        public bool Touched_Circular(MouseState touch)
        {
            currentState = buttonState.UP;
            texture = texture_up;
            if (OnButton((int)touch.X, (int)touch.Y))
            {
                currentState = buttonState.HOVER;
                texture = texture_hover;
                if (touch.LeftButton == ButtonState.Pressed)
                {
                    currentState = buttonState.DOWN;
                }
                return true;
            }
            return false;
        }
        public bool OnButton(int x, int y)
        {
            int radius = texture.Width / 2;
            float d = (float)Math.Sqrt(Math.Pow((x - rectangle.X), 2) + Math.Pow((y -
rectangle.Y), 2));
            if (d <= radius)
                return true;
            else
                return false;
```

```
        }
```

## Sprite.cs

This Class handles Sprite and Sprite Animation, which may or may not have a Farseer body attached to it. It has three constructors.

**//First Constructor takes Texture2D as a parameter and set the origin to the centre of the image.**

```
        public Sprite(Texture2D sprite)
        {
            Texture = sprite;
            Origin = new Vector2(sprite.Width / 2f, sprite.Height / 2f);
        }
```

**//Second Constructor takes Texture2D and Vector2(Origin) as parameters**

```
        public Sprite(Texture2D texture, Vector2 origin)
        {
            this.Texture = texture;
            this.Origin = origin;
        }
```

**//Third Constructor takes a bunch of parameters which handles spritesheet and sprite animation**

```
        public Sprite(Texture2D texture, Point frameSize, Point sheetSize, Point
currentFrame, float framesPerSec, int frameCount)
        {
            this.Texture = texture;
            this.frameSize = frameSize;
            this.sheetSize = sheetSize;
            this.currentFrame = currentFrame;
            this.TimePerFrame = (float)1/framesPerSec;
            this.frameCount = frameCount;
            this.Origin = new Vector2(frameSize.X / 2, frameSize.Y / 2);
        }
```

**//This method handles Sprite Animation**

```
        public void UpdateFrame(float elapsed, bool loop)
        {
            if (Paused)
                return;
            TotalElapsed += elapsed;
            if (TotalElapsed > TimePerFrame)
            {
                Frame++;
                Frame = Frame % frameCount;
                TotalElapsed -= TimePerFrame;
                currentFrame.X++;
                if (currentFrame.X > sheetSize.X)
                {
                    currentFrame.X = 0;
                    currentFrame.Y++;
                    if (currentFrame.Y > sheetSize.Y)
                        currentFrame.Y = 0;
                }
            }
            if (Frame == frameCount - 1 && !loop)
            {
```

```
                this.Pause();
            }
        }
```

**//This method draws the sprite with a Farseer body attached to it, simulating physics**

```
        public void DrawFrame(SpriteBatch batch, Body body, float rotation)
        {
            Rectangle sourcerect = new Rectangle(currentFrame.X * frameSize.X,
                                       currentFrame.Y * frameSize.Y,
                                       frameSize.X,
                                       frameSize.Y);
            batch.Draw(Texture, body.Position * Game1.MeterInPixels, sourcerect,
Color.White, rotation, Origin, 1f, SpriteEffects.None, 0f);
        }
```

**//This method draws the sprite with no Farseer body attached to it**

```
        public void DrawFrame(SpriteBatch batch, Vector2 pos)
        {
            Rectangle sourcerect = new Rectangle(currentFrame.X * frameSize.X,
                                       currentFrame.Y * frameSize.Y,
                                       frameSize.X,
                                       frameSize.Y);
            batch.Draw(Texture, pos, sourcerect, Color.White, 0f, Origin, 1f,
SpriteEffects.None, 0f);
        }
```

## ParticleEngine.cs

This class handles the particle system in the game. This class uses Particle.cs class to achieve its goals.

**Note**: Clouds.cs and CubeFallingSimulation.cs works in the similar fashion.

**//This method returns a particle**

```
        private Particle CreateParticle()
        {
            pvelocity = new Vector2((float)random.Next(0, 0), (float)random.Next(0,
10));
            Particle newparticle = new Particle(ptexture, new Vector2(random.Next(0,
800), -50), pvelocity, 0.1f, Color.White, 100);
            return newparticle;
        }
```

**//This method creates a List of Particle**

```
        public void UpdateParticles(int particleCount)
        {
            this.particleCount = particleCount;
            for (int i = 0; i < this.particleCount; i++)
            {
                particles.Add(CreateParticle());
            }
            DeleteParticles();
        }
```

**//This method updates the List of Particle and deletes them once their lifetime is less than zero**

```
private void DeleteParticles()
{
    for (int i = 0; i < particles.Count; i++)
    {
        particles[i].Update();
        if (particles[i].LifeTime <= 0)
            particles.RemoveAt(i);
    }
}
```

**//This method renders the particles on the screen**

```
public void DrawParticles(SpriteBatch spriteBatch)
{
    for (int i = 0; i < particles.Count; i++)
    {
        particles[i].Draw(spriteBatch);
    }
}
```

## Score.cs

This class handles score increment and decrement.

**//This block of code increases the score if any block falls off the platform and decreases score for each ball launched. It also plays the respective sound effect. Moreover, it removes the block from the memory once it falls off the platform. Thus, helping in memory management and optimizing the game.**

```
public int Update(List<Body> blocks)
{
    if (Level.currentMap == 5)
    {
        increaseBy = 15;
    }
    foreach (Body body in blocks)
    {
        if (CheckWallCollision(body))
        {
            return blocks.IndexOf(body);
        }
    }
    return -1;
}

public bool CheckWallCollision(Body body)
{
    Vector2 pos = body.Position * Game1.MeterInPixels;
    if (pos.Y > Game1.SCREEN_HEIGHT)
    {
        score += increaseBy;
        add.Play();
        return true;
    }
    return false;
```

```
        }
        public void scoreIncrease(List<Body> body)
        {
            int index = Update(body);
            if (index != -1)
            {
                body.RemoveAt(index);
                index = -1;
            }
        }

        public void scoreDecrease()
        {
            score += decreaseBy;
            sub.Play();
        }
```

## Launch_Pad.cs

This class handles the Spawning and releasing of the ball through States.

**//The Ball States**

```
    public enum GameState { reset, click, drag, release };
```

**//This method returns true if Mouse Click is registered within the Circular Launcher Pad**

```
        public bool OnLauncher(int x, int y)
        {
            int radius = launcher.Width / 2;
            float d = (float)Math.Sqrt(Math.Pow((x - launcherpos[Level.currentMap].X),
2) + Math.Pow((y - launcherpos[Level.currentMap].Y), 2));
            if (d <= radius)
                return true;
            else
                return false;
        }
```

**//This set of block shows how the ball spawns on Mouse Click, charges on Mouse Drag and releases on Mouse Release Events. Appropriate sound effects are played for each state.**

```
        public void Update(float elasped)
        {
            MouseState mouse = Mouse.GetState();
            ball.UpdateFrame(elasped, true);
            launcher_rotation += 0.01f;
            switch (currentState)
            {
                case GameState.reset:
                    startPointX = 0;
                    startPointY = 0;
                    scale = 0f;
                    force = 0;
                    rotation = 0f;
```

```
                    break;

                case GameState.release:
                    appliedforce = new Vector2((float)Math.Cos(rotation),
(float)Math.Sin(rotation)) * force / MeterInPixels;
                    ballBody.BodyType = BodyType.Dynamic;
                    launch.Play();

                    //ballBody.ApplyLinearImpulse(appliedforce * -1);
                    ballBody.ApplyForce(appliedforce * -1);

                    ballUsed = true;
                    ballReleased = false;
                    currentState = GameState.reset;
                    break;

                case GameState.click:
                    ini[random.Next(0, 5)].Play();
                    startPointX = mouse.X;
                    startPointY = mouse.Y;
                    ballBody.Position = new Vector2(startPointX, startPointY) /
MeterInPixels;
                    ballBody.BodyType = BodyType.Static;
                    break;

                case GameState.drag:
                    appliedforce = Vector2.Zero;

                    finalPointX = mouse.X;
                    finalPointY = mouse.Y;

                    int xDistance = (finalPointX - startPointX);
                    int yDistance = (finalPointY - startPointY);

                    float distance = (float)Math.Sqrt(Math.Pow((xDistance), 2) +
Math.Pow((yDistance), 2));
                    force = distance * 50f;
                    force = MathHelper.Clamp(force, 0 , force_limit);
                    rotation = (float)Math.Atan2(yDistance, xDistance);
                    scale = distance / ScaleToPixel;
                    ballReleased = true;
                    break;
            }

            if (mouse.LeftButton == ButtonState.Pressed)
            {
                if (startPointX == 0 && startPointY == 0 && OnLauncher(mouse.X,
mouse.Y))
                    currentState = GameState.click;
                else if (startPointX != 0 && startPointY != 0)
                    currentState = GameState.drag;
            }

            if (mouse.LeftButton == ButtonState.Released && ballReleased)
                currentState = GameState.release;
        }
```

## Level.cs

This class processes the map data from an .XML file and makes it playable, rendering it on the screen.

**//This set of block clears all the lists and reads the data from an XML file. It processes the data from the XML file and saves it in an multi-dimensional array**

```
        public void RemoveBodies(List<Body> body)
        {
            foreach (Body _body in body)
            {
                game.world.RemoveBody(_body);
            }
            body.Clear();
        }

        public void ReloadMap()
        {
            if (MediaPlayer.State == MediaState.Stopped)
                MediaPlayer.Play(level_music);

            if (mapReload)
            {
                //Delete Bodies
                RemoveBodies(YellowBlockBody);
                RemoveBodies(GreenTriangleBody);
                RemoveBodies(Hori_Rectangle_Body);
                RemoveBodies(Vert_Rectangle_Body);
                RemoveBodies(PlatformBody);

                numberColumns = Levels[currentMap].NumberColumns;
                numberRows = Levels[currentMap].NumberRows;

                MapTiles = new MapTileType[Levels[currentMap].NumberColumns,
Levels[currentMap].NumberRows];

                int u = 0;
                int v = 0;
                for (int i = 0; i < Levels[currentMap].Platform.Count; i++)
                {
                    u = Levels[currentMap].Platform[i].X;
                    v = Levels[currentMap].Platform[i].Y;
                    MapTiles[u, v] = MapTileType.MapPlatform;
                }


                u = 0;
                v = 0;
                for (int i = 0; i < Levels[currentMap].YellowBlock.Count; i++)
                {
                    u = Levels[currentMap].YellowBlock[i].X;
                    v = Levels[currentMap].YellowBlock[i].Y;
                    MapTiles[u, v] = MapTileType.MapYellowBlock;
                }


                u = 0;
```

```
            v = 0;
            for (int i = 0; i < Levels[currentMap].GreenTriangle.Count; i++)
            {
                u = Levels[currentMap].GreenTriangle[i].X;
                v = Levels[currentMap].GreenTriangle[i].Y;
                MapTiles[u, v] = MapTileType.MapGreenTriangle;
            }

            u = 0;
            v = 0;
            for (int i = 0; i < Levels[currentMap].Hori_Rectangle.Count; i++)
            {
                u = Levels[currentMap].Hori_Rectangle[i].X;
                v = Levels[currentMap].Hori_Rectangle[i].Y;
                MapTiles[u, v] = MapTileType.MapHori_Rectangle;
            }

            u = 0;
            v = 0;
            for (int i = 0; i < Levels[currentMap].Vert_Rectangle.Count; i++)
            {
                u = Levels[currentMap].Vert_Rectangle[i].X;
                v = Levels[currentMap].Vert_Rectangle[i].Y;
                MapTiles[u, v] = MapTileType.MapVert_Rectangle;
            }
            this.SetPos();
            mapReload = false;
        }
    }
```

**//This set of blocks processes the data from the multi-dimensional array and renders it on the screen**

```
    public Vector2 MapToWorld(int column, int row)
    {
        Vector2 screenPosition = new Vector2();
        if (InMap(column, row))
        {
            screenPosition.X = column * tileSize;
            screenPosition.Y = row * tileSize;
        }
        return screenPosition;
    }

    private bool InMap(int column, int row)
    {
        return (row >= 0 && row < numberRows && column >= 0 && column <
numberColumns);
    }

    public void SetPos()
    {
        for (int i = 0; i < numberRows; i++)
        {
            for (int j = 0; j < numberColumns; j++)
            {
                Vector2 tilePosition = MapToWorld(j, i);
                switch (MapTiles[j, i])
                {
                   case MapTileType.MapEmpty:
```

```
                        break;
                case MapTileType.MapPlatform:
                        Body tempPlatBody;
                        tempPlatBody = BodyFactory.CreateRectangle(game.world, 230
/ Game1.MeterInPixels, 25 / Game1.MeterInPixels, 0.5f);
                        tempPlatBody.BodyType = BodyType.Static;
                        tempPlatBody.Friction = 0.6f;
                        tempPlatBody.Position = tilePosition /
Game1.MeterInPixels;
                        PlatformBody.Add(tempPlatBody);
                        break;

                    case MapTileType.MapYellowBlock:
                        Body tempBody;
                        tempBody = BodyFactory.CreateRectangle(game.world, 25 /
Game1.MeterInPixels, 25 / Game1.MeterInPixels, 0.5f);
                        tempBody.BodyType = BodyType.Dynamic;
                        tempBody.Position = tilePosition / Game1.MeterInPixels;
                        YellowBlockBody.Add(tempBody);
                        break;

                    case MapTileType.MapHori_Rectangle:
                        Body tempBody2;
                        tempBody2 = BodyFactory.CreateRectangle(game.world, 40 /
Game1.MeterInPixels, 10 / Game1.MeterInPixels, 0.5f);
                        tempBody2.BodyType = BodyType.Dynamic;
                        tempBody2.Position = tilePosition / Game1.MeterInPixels;
                        Hori_Rectangle_Body.Add(tempBody2);
                        break;

                    case MapTileType.MapVert_Rectangle:
                        Body tempBody3;
                        tempBody3 = BodyFactory.CreateRectangle(game.world, 10 /
Game1.MeterInPixels, 40 / Game1.MeterInPixels, 0.5f);
                        tempBody3.BodyType = BodyType.Dynamic;
                        tempBody3.Position = tilePosition / Game1.MeterInPixels;
                        Vert_Rectangle_Body.Add(tempBody3);
                        break;

                    case MapTileType.MapGreenTriangle:
                        Body tempBody1;
                        Vertices vert = new Vertices();
                        vert.Add(new Vector2(0, -12.5f) / Game1.MeterInPixels);
                        vert.Add(new Vector2(12.5f, 12.5f) / Game1.MeterInPixels);
                        vert.Add(new Vector2(-12.5f, 12.5f) /
Game1.MeterInPixels);

                        tempBody1 = BodyFactory.CreatePolygon(game.world, vert,
0.5f);

                        tempBody1.BodyType = BodyType.Dynamic;
                        tempBody1.Position = tilePosition / Game1.MeterInPixels;
                        GreenTriangleBody.Add(tempBody1);
                        break;

                    default:
                        break;
                }
            }
        }
    }
```

```
        public void Draw(SpriteBatch spriteBatch)
        {
            //Instructions
            if (currentMap == 0)
            {
                spriteBatch.DrawString(Game1.font, "YOU GET POINTS FOR \n THROWING THE 
BLOCKS \n OFF THE STAGE", new Vector2(550, 260), Color.Black, 0f, Vector2.Zero, 0.6f, 
SpriteEffects.None, 0f);

                spriteBatch.DrawString(Game1.font, "THIS IS THE \"BALL SUMMONING 
AREA\" \n USAGE: PRESS, PULL, RELEASE TO LAUNCH BALL \n REMEMBER! BALLS COST POINTS", 
new Vector2(60, 400) , Color.Black, 0f, Vector2.Zero, 0.6f, SpriteEffects.None, 0f);
            }

            for (int i = 0; i < numberRows; i++)
            {
                for (int j = 0; j < numberColumns; j++)
                {
                    switch (MapTiles[j, i])
                    {
                        case MapTileType.MapPlatform:
                            foreach (Body body in PlatformBody)
                            {
                                spriteBatch.Draw(Platform.Texture, body.Position * 
Game1.MeterInPixels, null, Color.White, 0f, Platform.Origin, 1f, SpriteEffects.None, 
0f);
                            }
                            break;

                        case MapTileType.MapEmpty:
                            break;

                        case MapTileType.MapYellowBlock:
                            foreach (Body body in YellowBlockBody)
                            {
                                YellowBlock.DrawFrame(spriteBatch, body, 
body.Rotation);
                            }
                            break;

                        case MapTileType.MapHori_Rectangle:
                            foreach (Body body in Hori_Rectangle_Body)
                            {
                                Hori_Rectangle.DrawFrame(spriteBatch, body, 
body.Rotation);
                            }
                            break;

                        case MapTileType.MapVert_Rectangle:
                            foreach (Body body in Vert_Rectangle_Body)
                            {
                                Vert_Rectangle.DrawFrame(spriteBatch, body, 
body.Rotation);
                            }
                            break;

                        case MapTileType.MapGreenTriangle:
                            foreach (Body body in GreenTriangleBody)
                            {
```

```
                                spriteBatch.Draw(GreenTriangle.Texture, body.Position
* Game1.MeterInPixels, null, Color.White, body.Rotation, GreenTriangle.Origin, 1f,
SpriteEffects.None, 0f);
                            }
                            break;

                        default:
                            break;
                    }
                }
            }
        }
```