TPI 2020 - Mono Kong

Gama Ferreira Carvalho Tiago
I.FA-P3A
École d'informatique (CFPT-I)

09 Juin 2020

Table des matières

Chapitre 1

Monokong

1.1 Game

1.1.1 Game1.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using Microsoft.Xna.Framework;
10 using Microsoft.Xna.Framework.Graphics;
11 using DonkeyKong.States;
12 
13 namespace DonkeyKong
14 {
15     /// <summary>
16     /// This is the main type for your game.
17     /// </summary>
18     public class Game1 : Game
19     {
20         #region Variables
21 
22         GraphicsDeviceManager graphics;
23         SpriteBatch spriteBatch;
24 
25 
26         //Variables to use for the changing of states
27         private State _currentState;
28         private State _nextState;
29         #endregion
30 
31 
32         public Game1()
33         {
34             graphics = new GraphicsDeviceManager(this);
35             Content.RootDirectory = "Content";
36         }
37 
38         /// <summary>
39         /// Allows the game to perform any initialization it needs to ↩
               before starting to run.
40         /// This is where it can query for any required services and ↩
               load any non-graphic
41         /// related content.  Calling base.Initialize will enumerate ↩
               through any components
42         /// and initialize them as well.
43         /// </summary>
44         protected override void Initialize()
45         {
```

```
46
47              IsMouseVisible = false;
48              graphics.GraphicsProfile = GraphicsProfile.Reach;
49
50              graphics.PreferredBackBufferWidth = 900;
51              graphics.PreferredBackBufferHeight = 1000;
52
53
54              graphics.IsFullScreen = true;
55              graphics.ApplyChanges();
56              base.Initialize();
57          }
58
59          /// <summary>
60          /// LoadContent will be called once per game and is the place ↩
                to load
61          /// all of your content.
62          /// </summary>
63          protected override void LoadContent()
64          {
65              // Create a new SpriteBatch, which can be used to draw ↩
                    textures.
66              spriteBatch = new SpriteBatch(GraphicsDevice);
67              _currentState = new HomeMenu(this, graphics.GraphicsDevice, ↩
                    Content);
68          }
69
70
71          /// <summary>
72          /// Allows the game to run logic such as updating the world,
73          /// checking for collisions, gathering input, and playing audio.
74          /// </summary>
75          /// <param name="gameTime">Provides a snapshot of timing ↩
                values.</param>
76          protected override void Update(GameTime gameTime)
77          {
78              //If a next state is assigned
79              if (_nextState != null)
80              {
81                  _currentState = _nextState;
82                  _nextState = null;
83              }
84
85              _currentState.Update(gameTime);
86
87              base.Update(gameTime);
88          }
89
90          /// <summary>
91          /// This is called when the game should draw itself.
92          /// </summary>
93          /// <param name="gameTime">Provides a snapshot of timing ↩
                values.</param>
94          protected override void Draw(GameTime gameTime)
95          {
96              GraphicsDevice.Clear(Color.CornflowerBlue);
97              spriteBatch.Begin();
98              _currentState.Draw(gameTime, spriteBatch);
99
100
101             spriteBatch.End();
102             base.Draw(gameTime);
103         }
104         /// <summary>
105         /// Method to change in which state we are in
106         /// </summary>
107         /// <param name="state">Game or Menu state</param>
108         public void ChangeState(State state)
109         {
110             _nextState = state;
111         }
112     }
```

```
113 }
```

Listing 1.1 – ./DonkeyKong/DonkeyKong/Game1.cs

1.2 States

1.2.1 State.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using Microsoft.Xna.Framework;
10 using Microsoft.Xna.Framework.Content;
11 using Microsoft.Xna.Framework.Graphics;
12
13 namespace DonkeyKong.States
14 {
15
16     /// <summary>
17     /// The state class is a model for the creation of multiple states, ↩
       a "state" represents different parts of the program, for example ↩
       the menu is a state, and the game is another one.
18     /// Inspiration : https://github.com/Oyyou/MonoGame_Tutorials
19     /// </summary>
20     public abstract class State
21     {
22         #region Variables
23
24         protected ContentManager _content;
25
26         protected GraphicsDevice _graphicsDevice;
27
28         protected Game1 _game;
29
30         #endregion
31
32         #region Methods
33
34
35         public State(Game1 game, GraphicsDevice graphicsDevice, ↩
               ContentManager content)
36         {
37             _game = game;
38
39             _graphicsDevice = graphicsDevice;
40
41             _content = content;
42         }
43
44         public abstract void Update(GameTime gameTime);
45
46         public abstract void Draw(GameTime gameTime, SpriteBatch ↩
               spriteBatch);
47
48         #endregion
49     }
50 }
```

Listing 1.2 – ./DonkeyKong/DonkeyKong/States/State.cs

1.2.2 HomeMenu.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
```

```
6  * Version : 1.0
7  * Description : Recreation of the original Donkey Kong game by Nintendo
8  ***/
9
10 using System.Collections.Generic;
11 using Microsoft.Xna.Framework;
12 using Microsoft.Xna.Framework.Content;
13 using Microsoft.Xna.Framework.Graphics;
14 using Microsoft.Xna.Framework.Media;
15 using DonkeyKong.Sprites;
16 using DonkeyKong.Models;
17 using DonkeyKong.Managers;
18 using Microsoft.Xna.Framework.Input;
19 using DonkeyKong.Controls;
20 using Microsoft.Xna.Framework.Audio;
21 using DonkeyKong.GameComponents;
22
23 namespace DonkeyKong.States
24 {
25     /// <summary>
26     ///  The first page the user meets, they will be able to control ↩
           Mario and press any of the buttons to activate their ↩
           corresponding actions, such as playing the game, going to the ↩
           info page or exiting the game
27     ///  Inspiration : ↩
           https://github.com/Oyyou/MonoGame_Tutorials/tree/master/MonoGame_Tutorials/Tu
28     /// </summary>
29     class HomeMenu : State
30     {
31
32
33         private MenuButton playButton;
34         private MenuButton infoButton;
35         private MenuButton exitButton;
36
37         private Mario _mario;
38
39         private List<AnimatedSprite> _menuBarrels;
40
41         private AnimatedSprite _menuKong;
42
43         private Dictionary<string, Animation> animationsMovementMario;
44
45         private Song menuBackgroundMusic;
46         private Song gameBackgroundMusic;
47
48
49         private GenericSprite DKTitle;
50
51
52         public HomeMenu(Game1 game, GraphicsDevice graphicsDevice, ↩
               ContentManager content)
53           : base(game, graphicsDevice, content)
54         {
55
56             DKTitle = new GenericSprite(_game);
57             playButton = new MenuButton(_game);
58             infoButton = new MenuButton(_game);
59             exitButton = new MenuButton(_game);
60
61             LoadContent();
62             Initialize();
63
64
65         }
66
67         /// <summary>
68         /// LoadContent will be called once per game to load all the ↩
               content
69         /// </summary>
70         private void LoadContent()
71         {
```

```
72              animationsMovementMario = new Dictionary<string, Animation>()
73              {
74                  { "WalkRight", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkRight"),3)},
75                  { "WalkLeft", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkLeft"), 3)},
76                  { "WalkDown", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkRight"),3)},
77                  { "WalkUp", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkRight"), 3)},
78              };
79
80              Dictionary<string, SoundEffect> marioSOundEffects = new ↩
                    Dictionary<string, SoundEffect>()
81              {
82                  {"Walking", _content.Load<SoundEffect>( ↩
                        "Sounds/SoundEffects/walking") },
83                  {"Jump", _content.Load<SoundEffect>( ↩
                        "Sounds/SoundEffects/jump") },
84                  {"Climbing", _content.Load<SoundEffect>( ↩
                        "Sounds/SoundEffects/marioClimb") },
85              };
86
87              _mario = new Mario(_game, animationsMovementMario, ↩
                    marioSOundEffects, _graphicsDevice)
88              {
89                  Position = new Vector2(100, ↩
                        _graphicsDevice.Viewport.Height * 0.8f),
90                  Input = new Input()
91                  {
92                      Up = Keys.W,
93                      Down = Keys.S,
94                      Left = Keys.A,
95                      Right = Keys.D,
96                  }
97              };
98
99              var animationsMenuBarrels1 = new Dictionary<string, ↩
                    Animation>()
100             {
101                 { "Animated", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MenuBarrels"),2)},
102             };
103
104             var animationsMenuBarrels2 = new Dictionary<string, ↩
                    Animation>()
105             {
106                 { "Animated", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MenuBarrels"),2)},
107             };
108
109             _menuBarrels = new List<AnimatedSprite>()
110             {
111                 new AnimatedSprite(_game, animationsMenuBarrels1)
112                 {
113
114                     Position = new ↩
                            Vector2(_graphicsDevice.Viewport.Width * ↩
                            0.05f,_graphicsDevice.Viewport.Height * 0.35f),
115                 },
116                 new AnimatedSprite(_game, animationsMenuBarrels2)
117                 {
118
119                     Position = new ↩
                            Vector2(_graphicsDevice.Viewport.Width * ↩
                            0.82f,_graphicsDevice.Viewport.Height * 0.35f),
120                 },
121             };
122
123             var animationsMenuKong = new Dictionary<string, Animation>()
124             {
```

```
125                 { "Animated", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/KongIdleAnimation"),3)},
126             };
127 
128 
129             _menuKong = new AnimatedSprite(_game, animationsMenuKong);
130 
131             _menuKong.Position = new ↩
                    Vector2(_graphicsDevice.Viewport.Width / 2 - ↩
                    _menuKong._width / 2, _graphicsDevice.Viewport.Height * ↩
                    0.45f);
132 
133 
134             gameBackgroundMusic = _game.Content.Load<Song>( ↩
                    "Sounds/Music/gameMusic");
135             menuBackgroundMusic = _game.Content.Load<Song>( ↩
                    "Sounds/Music/menuMusic");
136 
137 
138             MediaPlayer.Play(menuBackgroundMusic);
139             MediaPlayer.Volume = 0.1f;
140             MediaPlayer.IsRepeating = true;
141 
142             DKTitle.LoadContent("Graphics/DKTitle");
143             playButton.LoadContent("Controls/PlayButton");
144             infoButton.LoadContent("Controls/InfoButton");
145             exitButton.LoadContent("Controls/ExitButton");
146         }
147 
148         /// <summary>
149         /// Ininializes all the necessary variables before the game starts
150         /// </summary>
151         private void Initialize()
152         {
153 
154             DKTitle.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width / 2 - ↩
                    DKTitle._texture.Width / 2, ↩
                    _graphicsDevice.Viewport.Height * 0.05f));
155             playButton.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width * 0.25f, ↩
                    _graphicsDevice.Viewport.Height * 0.75f));
156             playButton.Input = new Input()
157             {
158                 Action = new List<Keys>() { Keys.Space, Keys.F, Keys.G, ↩
                        Keys.H, Keys.V, Keys.B, Keys.N }
159             };
160             infoButton.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width * 0.45f, ↩
                    _graphicsDevice.Viewport.Height * 0.75f));
161             infoButton.Input = new Input()
162             {
163                 Action = new List<Keys>() { Keys.Space, Keys.F, Keys.G, ↩
                        Keys.H, Keys.V, Keys.B, Keys.N }
164             };
165             exitButton.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width * 0.65f, ↩
                    _graphicsDevice.Viewport.Height * 0.75f));
166             exitButton.Input = new Input()
167             {
168                 Action = new List<Keys>() { Keys.Space, Keys.F, Keys.G, ↩
                        Keys.H, Keys.V, Keys.B, Keys.N }
169             };
170 
171         }
172 
173         /// <summary>
174         /// Runs the state's logic, collisions, etc
175         /// </summary>
176         /// <param name="gameTime">Provides a snapshot of timing ↩
               values</param
177         public override void Update(GameTime gameTime)
```

```
178         {
179             DKTitle.Update(gameTime);
180 
181 
182             playButton.Update(gameTime, _mario.Hitbox);
183             infoButton.Update(gameTime, _mario.Hitbox);
184             exitButton.Update(gameTime, _mario.Hitbox);
185             ChangeStateConditions();
186 
187             foreach (var sprite in _menuBarrels)
188                 sprite.Update(gameTime);
189 
190             _mario.Update(gameTime);
191             _menuKong.Update(gameTime);
192 
193 
194         }
195 
196         /// <summary>
197         /// Conditions needed to go another state
198         /// </summary>
199         private void ChangeStateConditions()
200         {
201             if (playButton.ButtonPressed())
202             {
203                 MediaPlayer.Play(gameBackgroundMusic);
204                 MediaPlayer.Volume = 0.5f;
205                 _mario.StopAllSoundInstances();
206                 _game.ChangeState(new DonkeyKong(_game, ↩
                        _graphicsDevice, _content));
207 
208             }
209             else if (infoButton.ButtonPressed())
210             {
211                 _mario.StopAllSoundInstances();
212                 MediaPlayer.Stop();
213                 _game.ChangeState(new InfoPage(_game, _graphicsDevice, ↩
                        _content));
214 
215 
216             }
217             else if (exitButton.ButtonPressed())
218             {
219                 _game.Exit();
220             }
221         }
222 
223         /// <summary>
224         /// All objects are drawn here
225         /// </summary>
226         /// <param name="gameTime">Provides a snapshot of timing ↩
            values</param>
227         /// <param name="spriteBatch">Helper class for drawing strings ↩
            and sprites</param>
228         public override void Draw(GameTime gameTime, SpriteBatch ↩
            spriteBatch)
229         {
230             _graphicsDevice.Clear(Color.Black);
231 
232             DKTitle.Draw(spriteBatch);
233             playButton.Draw(spriteBatch);
234             infoButton.Draw(spriteBatch);
235             exitButton.Draw(spriteBatch);
236 
237             _mario.Draw(spriteBatch);
238 
239             foreach (var sprite in _menuBarrels)
240                 sprite.Draw(spriteBatch);
241 
242             _menuKong.Draw(spriteBatch);
243 
244         }
```

```
245
246         }
247 }
```

Listing 1.3 – ./DonkeyKong/DonkeyKong/States/HomeMenu.cs

1.2.3 InfoPage.cs

```
1     /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using System.Collections.Generic;
10 using DonkeyKong.Controls;
11 using DonkeyKong.GameComponents;
12 using DonkeyKong.Managers;
13 using DonkeyKong.Models;
14 using DonkeyKong.Sprites;
15 using Microsoft.Xna.Framework;
16 using Microsoft.Xna.Framework.Audio;
17 using Microsoft.Xna.Framework.Content;
18 using Microsoft.Xna.Framework.Graphics;
19 using Microsoft.Xna.Framework.Input;
20
21 namespace DonkeyKong.States
22 {
23     /// <summary>
24     /// Displays a page where a tutorial on how to play the game can be ↩
          found and also the credits
25     /// </summary>
26     class InfoPage : State
27     {
28
29         SpriteFont arcadeClassic;
30
31         GenericSprite DKTitle;
32
33         GenericSprite joystickRightLeft;
34         GenericSprite marioWalking;
35
36         GenericSprite joystickUpDown;
37         GenericSprite marioClimbing;
38
39         GenericSprite arcadeButtons;
40         GenericSprite marioJumping;
41
42         MenuButton goBackButton;
43
44         private Mario _mario;
45         Dictionary<string, Animation> animationsMovementMario;
46
47         public InfoPage(Game1 game, GraphicsDevice graphicsDevice, ↩
              ContentManager content) : base(game, graphicsDevice, content)
48         {
49
50             DKTitle = new GenericSprite(_game);
51
52             joystickRightLeft = new GenericSprite(_game);
53             marioWalking = new GenericSprite(_game);
54
55             joystickUpDown = new GenericSprite(_game);
56             marioClimbing = new GenericSprite(_game);
57
58             arcadeButtons = new GenericSprite(_game);
59             marioJumping = new GenericSprite(_game);
```

```
60
61              goBackButton = new MenuButton(_game);
62              LoadContent();
63              Initialize();
64
65          }
66
67          /// <summary>
68          /// LoadContent will be called once per game to load all the ↩
                content
69          /// </summary>
70          public void LoadContent()
71          {
72
73              arcadeClassic = _content.Load<SpriteFont>( "arcadeClassic");
74
75              DKTitle.LoadContent( "Graphics/DKTitle");
76
77              joystickRightLeft.LoadContent( "Graphics/JoystickRightLeft");
78              marioWalking.LoadContent( "Graphics/MarioMovingExample");
79
80              joystickUpDown.LoadContent( "Graphics/JoystickUpDown");
81              marioClimbing.LoadContent( "Graphics/MarioClimbingExample");
82
83              arcadeButtons.LoadContent( "Graphics/ArcadeButtons");
84              marioJumping.LoadContent( "Graphics/MarioJumpingExample");
85
86              goBackButton.LoadContent( "Controls/GoBackButton");
87
88
89              animationsMovementMario = new Dictionary<string, Animation>()
90              {
91                  { "WalkRight", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkRight"),3)},
92                  { "WalkLeft", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkLeft"), 3)},
93                  { "WalkDown", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkRight"),3)},
94                  { "WalkUp", new Animation(_content.Load<Texture2D>( ↩
                        "Graphics/Animations/MarioWalkRight"), 3)},
95              };
96
97              Dictionary<string, SoundEffect> marioSOundEffects = new ↩
                    Dictionary<string, SoundEffect>()
98              {
99                  {"Walking", _content.Load<SoundEffect>( ↩
                        "Sounds/SoundEffects/walking") },
100                 {"Jump", _content.Load<SoundEffect>( ↩
                        "Sounds/SoundEffects/jump") },
101                 {"Climbing", _content.Load<SoundEffect>( ↩
                        "Sounds/SoundEffects/marioClimb") },
102             };
103
104             _mario = new Mario(_game, animationsMovementMario, ↩
                    marioSOundEffects, _graphicsDevice)
105             {
106                 Position = new Vector2(_graphicsDevice.Viewport.Width * ↩
                        0.75f, _graphicsDevice.Viewport.Height * 0.85f),
107                 Input = new Input()
108                 {
109                     Up = Keys.W,
110                     Down = Keys.S,
111                     Left = Keys.A,
112                     Right = Keys.D,
113                 }
114             };
115         }
116
117         /// <summary>
118         /// Ininializes all the necessary variables before the game starts
119         /// </summary>
120         public void Initialize()
```

```
121         {
122             DKTitle.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width / 2 - ↩
                    DKTitle._texture.Width / 2, ↩
                    _graphicsDevice.Viewport.Height * 0.05f));
123             joystickRightLeft.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width * 0.1f, ↩
                    _graphicsDevice.Viewport.Height * 0.3f));
124             marioWalking.Initialize(new ↩
                    Vector2(joystickRightLeft._position.X + ↩
                    joystickRightLeft._texture.Width + ↩
                    _graphicsDevice.Viewport.Width * 0.1f, ↩
                    _graphicsDevice.Viewport.Height * 0.3f));
125
126             marioClimbing.Initialize(new Vector2( ↩
                    _graphicsDevice.Viewport.Width * 0.8f, ↩
                    _graphicsDevice.Viewport.Height * 0.3f));
127             joystickUpDown.Initialize(new ↩
                    Vector2(marioClimbing._position.X - joystickUpDown ↩
                    ._texture.Width - _graphicsDevice.Viewport.Width * 0.1f, ↩
                    _graphicsDevice.Viewport.Height * 0.3f - ↩
                    joystickUpDown._texture.Height / 7));
128
129             arcadeButtons.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width * 0.1f - ↩
                    arcadeButtons._texture.Width / 4, ↩
                    _graphicsDevice.Viewport.Height * 0.5f));
130             marioJumping.Initialize(new ↩
                    Vector2(arcadeButtons._position.X + ↩
                    arcadeButtons._texture.Width + ↩
                    _graphicsDevice.Viewport.Width * 0.1f, ↩
                    _graphicsDevice.Viewport.Height * 0.5f));
131
132             goBackButton.Initialize(new ↩
                    Vector2(_graphicsDevice.Viewport.Width * 0.85f, ↩
                    _graphicsDevice.Viewport.Height * 0.8f));
133             goBackButton.Input = new Input()
134             {
135                 Action = new List<Keys>() { Keys.Space, Keys.F, Keys.G, ↩
                        Keys.H, Keys.V, Keys.B, Keys.N }
136             };
137         }
138
139
140         /// <summary>
141         /// Runs the state's logic, collisions, etc
142         /// </summary>
143         /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
144         public override void Update(GameTime gameTime)
145         {
146             DKTitle.Update(gameTime);
147             joystickRightLeft.Update(gameTime);
148             marioWalking.Update(gameTime);
149
150             marioClimbing.Update(gameTime);
151             joystickUpDown.Update(gameTime);
152
153             arcadeButtons.Update(gameTime);
154             marioJumping.Update(gameTime);
155
156             _mario.Update(gameTime);
157
158
159             goBackButton.Update(gameTime, _mario.Hitbox);
160
161             ChangeStateConditions();
162         }
163
164         /// <summary>
165         /// Conditions needed to go another state
166         /// </summary>
```

```
167         private void ChangeStateConditions()
168         {
169             if (goBackButton.ButtonPressed())
170             {
171                 _mario.StopAllSoundInstances();
172                 _game.ChangeState(new HomeMenu(_game, _graphicsDevice, ↩
                        _content));
173             }
174         }
175 
176         /// <summary>
177         /// All objects are drawn here
178         /// </summary>
179         /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
180         /// <param name="spriteBatch">Helper class for drawing strings ↩
                and sprites</param>
181         public override void Draw(GameTime gameTime, SpriteBatch ↩
                spriteBatch)
182         {
183             spriteBatch.DrawString(arcadeClassic, "Original␣game␣by␣↩
                    nintendo", new Vector2(_graphicsDevice.Viewport.Width * ↩
                    0.05f , _graphicsDevice.Viewport.Height * 0.7f), ↩
                    Color.White);
184             spriteBatch.DrawString(arcadeClassic, "Recreated␣by␣Tiago␣↩
                    Gama", new Vector2(_graphicsDevice.Viewport.Width * ↩
                    0.05f, _graphicsDevice.Viewport.Height * 0.7f + ↩
                    arcadeClassic.MeasureString("Original␣game␣by␣↩
                    nintendo").Y), Color.White);
185 
186             _graphicsDevice.Clear(Color.Black);
187             DKTitle.Draw(spriteBatch);
188 
189             joystickRightLeft.Draw(spriteBatch);
190             marioWalking.Draw(spriteBatch);
191 
192             marioClimbing.Draw(spriteBatch);
193             joystickUpDown.Draw(spriteBatch);
194 
195             arcadeButtons.Draw(spriteBatch);
196             marioJumping.Draw(spriteBatch);
197 
198             goBackButton.Draw(spriteBatch);
199             _mario.Draw(spriteBatch);
200         }
201     }
202 }
```

Listing 1.4 – ./DonkeyKong/DonkeyKong/States/InfoPage.cs

1.2.4 DonkeyKong.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using DonkeyKong.GameComponents;
10 using DonkeyKong.Managers;
11 using DonkeyKong.Models;
12 using DonkeyKong.Sprites;
13 using Microsoft.Xna.Framework;
14 using Microsoft.Xna.Framework.Audio;
15 using Microsoft.Xna.Framework.Content;
16 using Microsoft.Xna.Framework.Graphics;
17 using Microsoft.Xna.Framework.Input;
```

```
18 using System;
19 using System.Collections.Generic;
20 using System.Linq;
21
22 namespace DonkeyKong.States
23 {
24     /// <summary>
25     /// Game state, this is where the game will take place
26     /// </summary>
27     class DonkeyKong : State
28     {
29
30         //Brick variables
31         Dictionary<string, List<Brick>> ground;
32         List<Brick> lineOfBricks;
33         //Examplary brick used to spawn all the others
34         Brick brick;
35
36         //Ladder variables
37         Ladder ladder;
38         List<Ladder> allLadders;
39
40         //Barrel variables
41         List<Barrel> allBarrels;
42         List<Barrel> toBeRemovedBarrels;
43
44         //Mario variables
45         Mario _mario;
46         Dictionary<string, Animation> animationsMovementMario;
47         Dictionary<string, SoundEffect> marioSoundEffects;
48         int _livesLeft;
49         List<GenericSprite> allLives;
50
51         //Kong variables
52         Kong _kong;
53         Dictionary<string, Animation> animationsKong;
54         Dictionary<string, SoundEffect> kongSoundEffects;
55
56         //Princess variables
57         Dictionary<string, Animation> princessAnimations;
58         AnimatedSprite _princess;
59
60         //Oil barrel variables
61         Dictionary<string, Animation> oilBarrelAnimations;
62         AnimatedSprite oilBarrel;
63
64         GenericSprite stackedBarrels;
65
66         //Fonts
67         SpriteFont arcadeClassic;
68         SpriteFont arcadeClassicBig;
69
70         //Game variables
71         GameTimer _gameTimer;
72         string score;
73         ScoreManager _scoreManager;
74         bool gameWon;
75         bool gameOver;
76         bool scoreSaved;
77         float timer;
78
79         //Game sounds
80         SoundEffect gameOverSound;
81         SoundEffectInstance gameOverInstance;
82         SoundEffect gameWonSound;
83         SoundEffectInstance gameWonInstance;
84
85
86         /// <param name="game">The game variable</param>
87         /// <param name="graphicsDevice">Information about the screen ↩
               used to display the game</param>
88         /// <param name="content"></param>
```

```
89          /// <param name="livesLeft">How many lives does mario have left</param>
90          /// <param name="gameDuration">How long has the game been going</param>
91          public DonkeyKong(Game1 game, GraphicsDevice graphicsDevice, ContentManager content, int livesLeft = 3, float gameDuration = 0f) : base(game, graphicsDevice, content)
92          {
93              //Initialize the game's variables at the start of the game
94              _livesLeft = livesLeft;
95              _gameTimer = new GameTimer(game, gameDuration);
96              _gameTimer.Started = true;
97              brick = new Brick(_game);
98              stackedBarrels = new GenericSprite(_game);
99              gameWon = false;
100             gameOver = false;
101             scoreSaved = false;
102
103             _scoreManager = ScoreManager.Load();
104             LoadContent();
105             Initialize();
106         }
107
108         /// <summary>
109         /// LoadContent will be called once per game to load all the content
110         /// </summary>
111         public void LoadContent()
112         {
113             brick.LoadContent( "Graphics/Ground");
114             stackedBarrels.LoadContent( "Graphics/StackedBarrels");
115
116             arcadeClassic = _content.Load<SpriteFont>( "arcadeClassic");
117             arcadeClassicBig = _content.Load<SpriteFont>( "arcadeClassicBig");
118             animationsMovementMario = new Dictionary<string, Animation>()
119             {
120                 { "WalkRight", new Animation(_content.Load<Texture2D>( "Graphics/Animations/MarioWalkRight"),3)},
121                 { "WalkLeft", new Animation(_content.Load<Texture2D>( "Graphics/Animations/MarioWalkLeft"), 3)},
122                 { "WalkDown", new Animation(_content.Load<Texture2D>( "Graphics/Animations/MarioWalkRight"),3)},
123                 { "Climb", new Animation(_content.Load<Texture2D>( "Graphics/Animations/MarioClimb"), 2)},
124                 { "JumpRight", new Animation(_content.Load<Texture2D>( "Graphics/Animations/MarioJumpingRight"), 2)},
125                 { "JumpLeft", new Animation(_content.Load<Texture2D>( "Graphics/Animations/MarioJumpingLeft"), 2)},
126             };
127
128             marioSoundEffects = new Dictionary<string, SoundEffect>()
129             {
130                 { "Walking", _content.Load<SoundEffect>( "Sounds/SoundEffects/walking") },
131                 { "Jump", _content.Load<SoundEffect>( "Sounds/SoundEffects/jump") },
132                 { "Climbing", _content.Load<SoundEffect>( "Sounds/SoundEffects/marioClimb") },
133             };
134
135
136
137             animationsKong = new Dictionary<string, Animation>()
138             {
139                 { "Idle", new Animation(_content.Load<Texture2D>( "Graphics/Animations/KongIdleAnimationSmall"),3)},
140                 { "GrabBarrel", new Animation(_content.Load<Texture2D>( "Graphics/Animations/KongBarrelAnimation"), 3)},
141             };
142
143             kongSoundEffects = new Dictionary<string, SoundEffect>()
```

```
            {
                {"Idle", _content.Load<SoundEffect>( ↩
                    "Sounds/SoundEffects/kongStomp") },
            };

            oilBarrelAnimations = new Dictionary<string, Animation>()
            {
                { "Animation", new Animation(_content.Load<Texture2D>( ↩
                    "Graphics/Animations/oilBarrelAnimation"), 2) }
            };

            princessAnimations = new Dictionary<string, Animation>()
            {
                { "Animation", new Animation(_content.Load<Texture2D>( ↩
                    "Graphics/Animations/PrincessAnimation"), 2) }
            };

            gameOverSound = _content.Load<SoundEffect>( ↩
                "Sounds/SoundEffects/gameOver");
            gameOverInstance = gameOverSound.CreateInstance();
            gameWonSound = _content.Load<SoundEffect>( ↩
                "Sounds/SoundEffects/gameWon");
            gameWonInstance = gameWonSound.CreateInstance();




        }


        /// <summary>
        /// Ininializes all the necessary variables before the game starts
        /// </summary>
        public void Initialize()
        {
            ground = new Dictionary<string, List<Brick>>();

            allLadders = new List<Ladder>();
            allLives = new List<GenericSprite>();

            brick.Initialize(new Vector2(_graphicsDevice.Viewport.Width ↩
                * 0f, _graphicsDevice.Viewport.Height - ↩
                brick._texture.Height));
            lineOfBricks = new List<Brick>();
            allBarrels = new List<Barrel>();
            toBeRemovedBarrels = new List<Barrel>();
            GroundLayoutSpawn();
            LadderSpawn();

            _mario = new Mario(_game, animationsMovementMario, ↩
                marioSoundEffects, _graphicsDevice, true)
            {
                Position = new Vector2(_graphicsDevice.Viewport.Width * ↩
                    0.1f, _graphicsDevice.Viewport.Height * 0.95f),
                Input = new Input()
                {
                    Up = Keys.W,
                    Down = Keys.S,
                    Left = Keys.A,
                    Right = Keys.D,
                    Action = new List<Keys>() { Keys.Space, Keys.F, ↩
                        Keys.G, Keys.H, Keys.V, Keys.B, Keys.N }
                },
                Speed = new Vector2(2f, 1.5f),

            };

            stackedBarrels._position = new ↩
                Vector2(_graphicsDevice.Viewport.Width * 0f + 1, ↩
                ground["Level5"][0]._position.Y - ↩
                stackedBarrels._texture.Height);

```

```
203             LivesLeftDisplay();
204
205             _kong = new Kong(_game, animationsKong, kongSoundEffects);
206             _kong.Position = new Vector2(stackedBarrels._texture.Width
                    + (stackedBarrels._texture.Width / 10),
                    ground["Level5"][0]._position.Y - _kong._height);
207
208             _princess = new AnimatedSprite(_game, princessAnimations);
209             _princess.Position = new
                    Vector2(ground["Level6"][0]._position.X,
                    ground["Level6"][0]._position.Y - _princess._height);
210
211             oilBarrel = new AnimatedSprite(_game, oilBarrelAnimations);
212             oilBarrel.Position = new
                    Vector2(_graphicsDevice.Viewport.Width * 0.01f,
                    brick._position.Y - oilBarrel._height);
213         }
214
215         /// <summary>
216         /// Creates the layout for game's platforms.
217         /// Will adapt to different screen sizes.
218         /// </summary>
219         private void GroundLayoutSpawn()
220         {
221             //First layer, generates a straight line of bricks
222             int nbBricksTotal = _graphicsDevice.Viewport.Width /
                    brick._texture.Width;
223             for (int i = 0; i <= nbBricksTotal; i++)
224             {
225                 Brick b = (Brick)brick.Clone();
226                 b._position.X = brick._position.X +
                        brick._texture.Width * i;
227                 b._position.Y = brick._position.Y;
228                 lineOfBricks.Add(b);
229             }
230
231             ground.Add("Level0", lineOfBricks);
232             lineOfBricks = new List<Brick>();
233
234             //Layer 1 to 5
235             //Gets the amount of bricks needed to fill the height wise
236             int bricksHeight = brick._texture.Height * 7;
237             int spaceLeft = _graphicsDevice.Viewport.Height -
                    bricksHeight;
238             int spacePerBrick = spaceLeft / 7;
239
240             //Inclination to be used for the platforms
241             int totalInclination =
                    (int)(_graphicsDevice.Viewport.Height * 0.05);
242
243             //Variable to alternate between sticking to the right or left
244             bool stickToTheRight = false;
245
246             //How many bricks can (85% of) a line fit
247             int nbBricksPerLine = (int)(_graphicsDevice.Viewport.Width
                    * 0.85f) / brick._texture.Width;
248             //Inclination for each brick individually
249             int inclinationPerBlock = totalInclination / nbBricksPerLine;
250
251             //Generates the next 5 platforms
252             for (int i = 1; i < 6; i++)
253             {
254                 for (int j = 0; j <= nbBricksPerLine; j++)
255                 {
256                     //Clone a new brick
257                     Brick b = (Brick)brick.Clone();
258                     //If we're sticking to the right, start from the
                            right of the screen and substract each block added
259                     if (stickToTheRight)
260                     {
261                         b._position.X = _graphicsDevice.Viewport.Width
                                - brick._texture.Width * (j + 1);
```

```
262                     }
263                     else // Start from 0 and add each brick
264                     {
265                         b._position.X = brick._position.X + ↩
                               brick._texture.Width * j;
266 
267                     }
268 
269                     if (i == 5) // The 5th platform is special, the ↩
                           inclination isnt for every brick, but only the ↩
                           last few
270                     {
271                         //1.3 was the number I decided after playing a ↩
                               bit with the numbers, no other specific reason
272                         int inclinedBricks = (int)(nbBricksPerLine / ↩
                               1.3f);
273 
274                         if (j > inclinedBricks) // If we've reached the ↩
                               bricks to incline the incline them
275                         {
276                             int totalInclinationFirstLine = ↩
                                   (int)(_graphicsDevice.Viewport.Height * ↩
                                   0.03);
277                             int inclinationPerBlockFirstLine = ↩
                                   totalInclinationFirstLine / inclinedBricks;
278 
279                             b._position.Y = brick._position.Y - ↩
                                   brick._texture.Height * i - ↩
                                   spacePerBrick * i + ↩
                                   inclinationPerBlockFirstLine * (j - ↩
                                   inclinedBricks);
280                         }
281                         else // Otherwise just spawn them in a straight ↩
                               line
282                         {
283                             b._position.Y = brick._position.Y - ↩
                                   brick._texture.Height * i - ↩
                                   spacePerBrick * i;
284                         }
285 
286                     }
287                     else //If its not the 5th just incline the entire line
288                     {
289                         b._position.Y = brick._position.Y - ↩
                               brick._texture.Height * i - spacePerBrick * ↩
                               i + inclinationPerBlock * j;
290                     }
291                     lineOfBricks.Add(b);
292                 }
293 
294                 ground.Add("Level" + i.ToString(), lineOfBricks);
295                 lineOfBricks = new List<Brick>();
296 
297                 stickToTheRight = !stickToTheRight;
298             }
299 
300             //Layer 6
301             //For the platform get 1/4 of the total bricks per screen
302             //And create a straight platform for the princess to sit, ↩
                   this were the game will end if Mario reaches it
303             int oneFourthOfTotalBricksPerLine = nbBricksTotal / 4;
304 
305             for (int i = 0; i < oneFourthOfTotalBricksPerLine; i++)
306             {
307                 Brick b = (Brick)brick.Clone();
308                 b._position.X = _graphicsDevice.Viewport.Width * 0.4f + ↩
                       brick._position.X + brick._texture.Width * i;
309                 b._position.Y = _graphicsDevice.Viewport.Height * 0.15f;
310                 lineOfBricks.Add(b);
311             }
312             ground.Add("Level6", lineOfBricks);
313         }
```

```
314
315         /// <summary>
316         /// Spawns all the ladders, each ladder will be spawned in the 4th (counting from the end of a line) brick of each line and go down untill it meets a brick.
317         /// </summary>
318         private void LadderSpawn()
319         {
320             //Go from the last line to the first
321             for (int i = 6; i > 0; i--)
322             {
323                 ladder = new Ladder(_game);
324                 ladder.LoadContent("Graphics/Ladder");
325 
326                 Brick closestBrick = null;
327                 //Start the minimum with the max distance possible
328                 float minDist = _graphicsDevice.Viewport.Width;
329                 int startingBrickListPos;
330 
331                 //If its the princess line use the last brick instead of the 4th counting from the end
332                 if (i == 6)
333                 {
334                     startingBrickListPos = ground["Level" + i.ToString()].Count - 1;
335                 }
336                 else
337                 {
338                     startingBrickListPos = ground["Level" + i.ToString()].Count - 4;
339                 }
340 
341                 //For each brick in the line below
342                 foreach (Brick b in ground["Level" + (i - 1).ToString()])
343                 {
344                     //Get the difference between their X positions
345                     float dist = Math.Abs(ground["Level" + i.ToString()][startingBrickListPos]._position.X - b._position.X);
346 
347                     //And if its smaller than the current minimum distance
348                     if (dist < minDist)
349                     {
350                         //Set the new min distance and set the closest brick
351                         minDist = dist;
352                         closestBrick = b;
353                     }
354                 }
355                 //Get the vertical space between both bricks
356                 float spaceBetweenPlatforms = brick._texture.Height + closestBrick._position.Y - ground["Level" + i.ToString()][startingBrickListPos]._position.Y;
357                 //Divide it by the ladder height, to get how many mini ladders we need to form a complete ladder
358                 ladder.NbSpritesInStack = (int)(spaceBetweenPlatforms / ladder._texture.Height);
359                 //Place the ladder at the starting brick position
360                 ladder._position = new Vector2(ground["Level" + i.ToString()][startingBrickListPos]._position.X, ground["Level" + i.ToString()][startingBrickListPos]._position.Y);
361                 //Add it to the ladder list
362                 allLadders.Add(ladder);
363             }
364 
365 
366         }
367 
368         /// <summary>
369         /// Spawns a barrel when conditions are met
370         /// </summary>
```

```
371         private void BarrelSpawn()
372         {
373             if (_kong.CanSpawnBarrel())
374             {
375                 //Create a new animation for each barrel, otherwise ↩
                        animations will speed up
376                 Dictionary<string, Animation> bAnimation = new ↩
                        Dictionary<string, Animation>()
377                 { { "Animation", new ↩
                        Animation(_content.Load<Texture2D>("Graphics/Animations/barrelAni
                        4)}};
378 
379                 Barrel ba = new Barrel(_game, bAnimation, _graphicsDevice)
380                 {
381                     Position = new Vector2(_kong.hitbox.Right, ↩
                            _kong.Position.Y +  _kong.hitbox.Height / 2),
382                     Speed = new Vector2(4f, 3f)
383                 };
384 
385                 //Adds the barrel to the barrel list
386                 allBarrels.Add(ba);
387 
388             }
389 
390         }
391 
392         /// <summary>
393         /// Updates the visual mario lives in the upper left corner
394         /// </summary>
395         private void LivesLeftDisplay()
396         {
397             for (int i = 0; i < _livesLeft; i++)
398             {
399                 GenericSprite lifeSprite = new GenericSprite(_game);
400                 lifeSprite.LoadContent("Graphics/marioLife");
401                 lifeSprite._position = new ↩
                        Vector2(stackedBarrels._position.X + ↩
                        lifeSprite._texture.Width * i, 0);
402                 allLives.Add(lifeSprite);
403             }
404         }
405 
406         /// <summary>
407         /// Runs all the game logic, all the collisions, animations, ↩
                sounds, etc
408         /// </summary>
409         /// <param name="gameTime"></param>
410         public override void Update(GameTime gameTime)
411         {
412             //If the game isnt over
413             if (!gameOver)
414             {
415                 //Pressing any of the upper arcade buttons will return ↩
                        to the menu
416                 if (Keyboard.GetState().IsKeyDown(Keys.D6) ||
417                     Keyboard.GetState().IsKeyDown(Keys.D7) ||
418                     Keyboard.GetState().IsKeyDown(Keys.D8) ||
419                     Keyboard.GetState().IsKeyDown(Keys.D9) ||
420                     Keyboard.GetState().IsKeyDown(Keys.D0) ||
421                     Keyboard.GetState().IsKeyDown(Keys.Escape))
422                 {
423                     GoBackToMenu();
424                 }
425 
426                 //Update the game timer which is used for the score
427                 _gameTimer.Update(gameTime);
428                 score = _gameTimer.Text;
429                 int scoreLength = score.Length;
430                 //All display at least 4 digits
431                 for (int i = 0; i < 4 - scoreLength; i++)
432                 {
433                     score = "0" + score;
```

```
434                 }
435
436
437                 brick.Update(gameTime);
438                 foreach (List<Brick> lb in ground.Values)
439                 {
440                     foreach (Brick b in lb)
441                     {
442                         b.Update(gameTime);
443                     }
444                 }
445
446
447                 BarrelLogic(gameTime);
448
449                 foreach (Ladder l in allLadders)
450                 {
451                     l.Update(gameTime);
452                 }
453
454                 stackedBarrels.Update(gameTime);
455                 _princess.Update(gameTime);
456
457                 foreach (GenericSprite marioLives in allLives)
458                 {
459                     marioLives.Update(gameTime);
460                 }
461
462                 _kong.Update(gameTime);
463                 _mario.Update(gameTime, ground, allLadders, allBarrels);
464
465                 MarioBarrelCollision();
466
467                 WinCondition();
468             }
469             else //If the game is over
470             {
471                 if (gameWon) //And you won it
472                 {
473                     //Save your score unless you already did it
474                     if (!scoreSaved)
475                     {
476                         _scoreManager.Add(new Score()
477                         {
478                             PlayerName = "NONAME",
479                             Value = score,
480                         });
481                         ScoreManager.Save(_scoreManager);
482                         scoreSaved = true;
483                     }
484
485                     //Play the win game music
486                     gameWonInstance.Play();
487                 }
488                 else //Or you lost it
489                 {
490                     //Play the lose game music
491                     gameOverInstance.Play();
492                 }
493
494                 //Start a timer and when it reaches 0 go back to the menu
495                 //Timer's length is equal to the audio length
496                 float elapsed = ↩
                        (float)gameTime.ElapsedGameTime.TotalSeconds;
497                 timer -= elapsed;
498                 if (timer < 0)
499                 {
500                     GoBackToMenu();
501                 }
502             }
503
504         }
```

```
505
506         /// <summary>
507         /// Checks whenever mario hits a barrel, if he has enough lives to keep playing or if it is a game over.
508         /// </summary>
509         private void MarioBarrelCollision()
510         {
511             if (_mario.IsMarioDead() == true)
512             {
513                 _livesLeft--;
514                 if (_livesLeft > 0)
515                 {
516                     _game.ChangeState(new DonkeyKong(_game, _graphicsDevice, _content, _livesLeft, Convert.ToInt32(score)));
517                 }
518                 else
519                 {
520                     gameOver = true;
521                     //The 3s represent the losing music audio length
522                     timer = 3;
523                 }
524 
525             }
526         }
527 
528         /// <summary>
529         /// Checkes for when Mario reaches the princess and wins
530         /// </summary>
531         private void WinCondition()
532         {
533             if (_mario.Hitbox.Intersects(_princess.hitbox))
534             {
535                 gameOver = true;
536                 gameWon = true;
537                 //The 3s represent the winning music audio length
538                 timer = 5.5f;//Audio length
539             }
540         }
541 
542         /// <summary>
543         /// Changes states and goes to the menu
544         /// </summary>
545         private void GoBackToMenu()
546         {
547             //Stops the sound from carrying over to the menu;
548             _mario.StopAllSoundInstances();
549             _game.ChangeState(new HomeMenu(_game, _graphicsDevice, _content));
550         }
551 
552         /// <summary>
553         /// All barrels updates are here and checks for when a barrel needs to be removed from the game.
554         /// </summary>
555         /// <param name="gameTime">Provides a snapshot of timing values</param>
556         private void BarrelLogic(GameTime gameTime)
557         {
558             oilBarrel.Update(gameTime);
559 
560             foreach (Barrel ba in allBarrels)
561             {
562                 ba.Update(gameTime, ground);
563 
564                 //If the barrel reaches the oil barrel at the end
565                 if (ba.IsCollidingWithOilBarrel(oilBarrel))
566                 {
567                     //Add it to the being removed list
568                     toBeRemovedBarrels.Add(ba);
569 
570                 }
```

```
571             }
572
573             //Remove all barrels who reached the end from the barrel list
574             foreach (Barrel b in toBeRemovedBarrels)
575             {
576                 allBarrels.Remove(b);
577             }
578
579             toBeRemovedBarrels.Clear();
580
581             BarrelSpawn();
582         }
583
584         /// <summary>
585         /// All game components are drawn here.
586         /// </summary>
587         /// <param name="gameTime">Provides a snapshot of timing values</param>
588         /// <param name="spriteBatch">Helper class for drawing strings and sprites</param>
589         public override void Draw(GameTime gameTime, SpriteBatch spriteBatch)
590         {
591             _graphicsDevice.Clear(Color.Black);
592             brick.Draw(spriteBatch);
593
594             foreach (Ladder l in allLadders)
595             {
596                 l.Draw(spriteBatch);
597             }
598
599
600             foreach (List<Brick> lb in ground.Values)
601             {
602                 foreach (Brick b in lb)
603                 {
604                     b.Draw(spriteBatch);
605                 }
606             }
607
608             foreach (Barrel ba in allBarrels)
609             {
610                 ba.Draw(spriteBatch);
611             }
612             _princess.Draw(spriteBatch);
613             stackedBarrels.Draw(spriteBatch);
614
615             foreach (GenericSprite gs in allLives)
616             {
617                 gs.Draw(spriteBatch);
618             }
619             oilBarrel.Draw(spriteBatch);
620             _kong.Draw(spriteBatch);
621             _mario.Draw(spriteBatch);
622
623
624             spriteBatch.DrawString(arcadeClassic, "HIGHSCORE␣␣" + _scoreManager.Highscores[0].Value, new Vector2(_graphicsDevice.Viewport.Width - arcadeClassic.MeasureString("HIGHSCORE␣␣" + _scoreManager.Highscores[0].Value).X, 0), Color.White);
625             spriteBatch.DrawString(arcadeClassic, "SCORE␣␣" + score, new Vector2(_graphicsDevice.Viewport.Width - arcadeClassic.MeasureString("SCORE␣␣" + score).X, arcadeClassic.MeasureString("HIGHSCORE␣␣" + _scoreManager.Highscores.Select(c => c.Value)).Y), Color.White);
626             EndGameDisplays(spriteBatch);
627
628         }
629
630         /// <summary>
```

```
631         /// When the game is over either display a You Win + the score ↩
                or display Game Over
632         /// </summary>
633         /// <param name="spriteBatch"></param>
634         private void EndGameDisplays(SpriteBatch spriteBatch)
635         {
636             if (gameWon)
637             {
638                 spriteBatch.DrawString(arcadeClassicBig, "YOU␣WIN", new ↩
                        Vector2(_graphicsDevice.Viewport.Width / 2 - ↩
                        arcadeClassicBig.MeasureString("YOU␣WIN").X / 2, ↩
                        _graphicsDevice.Viewport.Height * 0.3f), Color.Red);
639                 spriteBatch.DrawString(arcadeClassicBig, score, new ↩
                        Vector2(_graphicsDevice.Viewport.Width / 2 - ↩
                        arcadeClassicBig.MeasureString(score).X / 2, ↩
                        _graphicsDevice.Viewport.Height * 0.3f + ↩
                        arcadeClassicBig.MeasureString("YOU␣WIN").Y), ↩
                        Color.White);
640             }
641             else if (gameOver)
642             {
643                 spriteBatch.DrawString(arcadeClassicBig, "GAME␣OVER", ↩
                        new Vector2(_graphicsDevice.Viewport.Width / 2 - ↩
                        arcadeClassicBig.MeasureString("GAME␣OVER").X / 2, ↩
                        _graphicsDevice.Viewport.Height * 0.3f), Color.Red);
644             }
645         }
646     }
647 }
```

Listing 1.5 – ./DonkeyKong/DonkeyKong/States/DonkeyKong.cs

1.3 Sprites

1.3.1 GenericSprite.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using Microsoft.Xna.Framework;
10 using Microsoft.Xna.Framework.Graphics;
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Text;
15 using System.Threading.Tasks;
16
17 namespace DonkeyKong.Sprites
18 {
19     /// <summary>
20     /// A sprite with generic attributes such as a position, hitbox and ↩
          a texture, mainly meant to be used for heritage
21     /// </summary>
22     class GenericSprite
23     {
24         #region Variables
25         protected Game _game;
26
27         public Vector2 _position;
28
29         public Texture2D _texture;
30         public Rectangle hitbox;
31
32
33         #endregion
34
35         #region Constructor + Initialize + LoadContent
36         /// <summary>
37         /// Constructor, requires the game to be able to load content
38         /// Usually not used by anyone but its children
39         /// </summary>
40         public GenericSprite(Game game)
41         {
42             _game = game;
43         }
44
45         /// <summary>
46         /// Initializes the object in a certain position
47         /// </summary>
48         /// <param name="position">Position in the screen</param>
49         public virtual void Initialize(Vector2 position)
50         {
51             _position = position;
52         }
53
54         /// <summary>
55         /// Loads the file texture into a variable
56         /// </summary>
57         /// <param name="texture">Name of the file</param>
58         public void LoadContent(string texture)
59         {
60             _texture = _game.Content.Load<Texture2D>(texture);
61         }
62
63         #endregion
64
65         #region Update + Draw
66
```

```
67          /// <summary>
68          /// Updates the position and hitbox of the sprite
69          /// </summary>
70          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
71          public virtual void Update(GameTime gameTime)
72          {
73
74
75              hitbox = new Rectangle(
76                      (int)_position.X,
77                      (int)_position.Y,
78                      _texture.Width,
79                      _texture.Height);
80          }
81          /// <summary>
82          /// Draws the sprite with a texture an hitbox and keeps its ↩
                original color
83          /// </summary>
84          /// <param name="spriteBatch">Helper class for drawing strings ↩
                and sprites</param>
85          public virtual void Draw(SpriteBatch spriteBatch)
86          {
87              //Draw the sprite
88              spriteBatch.Draw(_texture, hitbox, Color.White);
89          }
90
91          #endregion
92      }
93  }
```

Listing 1.6 – ./DonkeyKong/DonkeyKong/Sprites/GenericSprite.cs

1.3.2 AnimatedSprite.cs

```
1     /***
2   * Program : DonkeyKong
3   * Author : Tiago Gama
4   * Project : TPI 2020
5   * Date : 25.05.2020 - 09.06.2020
6   * Version : 1.0
7   * Description : Recreation of the original Donkey Kong game by Nintendo
8   ***/
9   using DonkeyKong.Managers;
10  using DonkeyKong.Models;
11  using Microsoft.Xna.Framework;
12  using Microsoft.Xna.Framework.Audio;
13  using Microsoft.Xna.Framework.Graphics;
14  using Microsoft.Xna.Framework.Input;
15  using System;
16  using System.Collections.Generic;
17  using System.Linq;
18  using System.Text;
19  using System.Threading.Tasks;
20
21  namespace DonkeyKong.Sprites
22  {
23
24      /// <summary>
25      /// A sprite with animations that cannot be moved
26      /// Inspiration : ↩
            https://github.com/Oyyou/MonoGame_Tutorials/tree/master/MonoGame_Tutorials/Tu
27      /// </summary>
28      class AnimatedSprite : GenericSprite
29      {
30          #region Fields
31
32          public AnimationManager _animationManager;
33
```

```
34          public Dictionary<string, Animation> _animations;
35
36
37
38          #endregion
39
40          #region Properties
41
42
43          /// <summary>
44          /// When setting the position property, also set the animation position
45          /// </summary>
46          public Vector2 Position
47          {
48              get { return _position; }
49              set
50              {
51                  _position = value;
52
53                  if (_animationManager != null)
54                      _animationManager.Position = _position;
55              }
56          }
57
58          public int _height;
59          public int _width;
60
61
62          public Vector2 Speed = new Vector2(2f, 2f);
63
64          public Vector2 Velocity;
65
66          #endregion
67
68          #region Methods
69
70
71          public AnimatedSprite(Game game, Dictionary<string, Animation> animations) : base(game)
72          {
73              _animations = animations;
74              _animationManager = new AnimationManager(_animations.First().Value);
75
76              //For the initial animatedSprite position, in the beginning the hitbox will not be set, so we use these
77              _height = _animations.First().Value.FrameHeight;
78              _width = _animations.First().Value.FrameWidth;
79          }
80
81          public AnimatedSprite(Game game, Texture2D texture) : base(game)
82          {
83              _texture = texture;
84          }
85
86
87          /// <summary>
88          /// Runs the position and hitbox logic, also updates the animation manager
89          /// </summary>
90          /// <param name="gameTime">Provides a snapshot of timing values</param>
91          public override void Update(GameTime gameTime)
92          {
93
94              _animationManager.Update(gameTime);
95              hitbox = new Rectangle(
96                      (int)_position.X,
97                      (int)_position.Y,
98                      _animations.First().Value.FrameWidth,
99                      _animations.First().Value.FrameHeight);
```

```
100
101
102
103             Position += Velocity;
104             Velocity = Vector2.Zero;
105         }
106
107         /// <summary>
108         /// Draws the animated sprite
109         /// </summary>
110         /// <param name="spriteBatch">Helper class for drawing strings ↩
                and sprites</param>
111         public override void Draw(SpriteBatch spriteBatch)
112         {
113
114             if (_texture != null)
115                 spriteBatch.Draw(_texture, Position, Color.White);
116             else if (_animationManager != null)
117                 _animationManager.Draw(spriteBatch);
118
119         }
120         #endregion
121     }
122 }
```

Listing 1.7 – ./DonkeyKong/DonkeyKong/Sprites/AnimatedSprite.cs

1.3.3 MovingAnimatedSprite.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using DonkeyKong.Managers;
10 using DonkeyKong.Models;
11 using Microsoft.Xna.Framework;
12 using Microsoft.Xna.Framework.Graphics;
13 using System.Collections.Generic;
14
15 namespace DonkeyKong.Sprites
16 {
17     /// <summary>
18     /// A sprite with animations that can be moved
19     /// Inspiration : ↩
          https://github.com/Oyyou/MonoGame_Tutorials/tree/master/MonoGame_Tutorials/Tu
20     /// </summary>
21     class MovingAnimatedSprite : AnimatedSprite
22     {
23         public Input Input;
24
25         public MovingAnimatedSprite(Game game, Dictionary<string, ↩
              Animation> animations):base(game, animations)
26         {
27
28         }
29
30         public MovingAnimatedSprite(Game game, Texture2D ↩
              texture):base(game, texture)
31         {
32
33         }
34
35         /// <summary>
36         /// Class to be overriden, sets the current animation and sound
37         /// </summary>
```

```
38          protected virtual void SetAnimationsAndSounds(){}
39
40
41          /// <summary>
42          /// Basic logic for a moving animated sprite, update its ↩
                animations, change them when necessary and update its position
43          /// </summary>
44          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
45          public virtual void Update(GameTime gameTime)
46          {
47              SetAnimationsAndSounds();
48
49              _animationManager.Update(gameTime);
50
51              Position += Velocity;
52              Velocity = Vector2.Zero;
53
54          }
55      }
56  }
```

Listing 1.8 – ./DonkeyKong/DonkeyKong/Sprites/MovingAnimatedSprite.cs

1.4 Models

1.4.1 Animation.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using Microsoft.Xna.Framework.Graphics;
10
11 namespace DonkeyKong.Models
12 {
13     /// <summary>
14     /// This is an animation model, it contains all the variables ↩
          necessary to manage an animation.
15     /// Inspiration : ↩
          https://github.com/Oyyou/MonoGame_Tutorials/blob/master/MonoGame_Tutorials/Tu
16     /// </summary>
17     class Animation
18     {
19         public int CurrentFrame { get; set; }
20
21         public int FrameCount { get; private set; }
22
23         public int FrameHeight { get { return Texture.Height; } }
24
25         public float FrameSpeed { get; set; }
26
27         public int FrameWidth { get { return Texture.Width / ↩
              FrameCount; } }
28
29         public bool IsLooping { get; set; }
30
31         public Texture2D Texture { get; private set; }
32
33         public Animation(Texture2D texture, int frameCount)
34         {
35             Texture = texture;
36
37             FrameCount = frameCount;
38
39             IsLooping = true;
40
41             //The higher it is, the slower the animation becomes
42             FrameSpeed = 0.15f;
43         }
44     }
45 }
```

Listing 1.9 – ./DonkeyKong/DonkeyKong/Models/Animation.cs

1.4.2 Score.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using System;
10 using System.Collections.Generic;
11 using System.Linq;
```

```
12 using System.Text;
13 using System.Threading.Tasks;
14
15 namespace DonkeyKong.Models
16 {
17     /// <summary>
18     /// This is a score model, it contains all the variables necessary ↩
          to manage a score.
19     /// Inspiration : ↩
          https://github.com/Oyyou/MonoGame_Tutorials/tree/master/MonoGame_Tutorials/Tu
20     /// </summary>
21     public class Score
22     {
23         public string PlayerName { get; set; }
24         public string Value { get; set; }
25     }
26 }
```

Listing 1.10 – ./DonkeyKong/DonkeyKong/Models/Score.cs

1.5 Managers

1.5.1 AnimationManager.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9
10 using DonkeyKong.Models;
11 using Microsoft.Xna.Framework;
12 using Microsoft.Xna.Framework.Graphics;
13
14 namespace DonkeyKong.Managers
15 {
16     /// <summary>
17     /// This class contains all the logic for the animations and also ↩
           displays it
18     /// Inspiration : https://github.com/Oyyou/MonoGame_Tutorials
19     /// </summary>
20     class AnimationManager
21     {
22         public Animation _animation;
23
24         private float _timer;
25
26         public Vector2 Position { get; set; }
27
28         /// <summary>
29         /// Starts the animation manager with one animation to manage
30         /// </summary>
31         /// <param name="animation">A texture with multiple frames to ↩
              be cycled through</param>
32         public AnimationManager(Animation animation)
33         {
34             _animation = animation;
35         }
36
37         /// <summary>
38         /// Runs all the animation logic, such as how long each frame ↩
              appears and the cycling of frames
39         /// </summary>
40         /// <param name="gameTime"></param>
41         public void Update(GameTime gameTime)
42         {
43             _timer += (float)gameTime.ElapsedGameTime.TotalSeconds;
44
45             if (_timer > _animation.FrameSpeed)
46             {
47                 _timer = 0f;
48
49                 _animation.CurrentFrame++;
50
51                 if (_animation.CurrentFrame >= _animation.FrameCount)
52                     _animation.CurrentFrame = 0;
53             }
54         }
55
56
57         /// <summary>
58         /// If the animations isnt already playing, play it and at the ↩
              end stop doing it, so it doesnt do it forever.
59         /// </summary>
60         /// <param name="animation">A texture with multiple frames to ↩
              be cycled through</param>
61         public void Play(Animation animation)
62         {
```

```
63              if (_animation == animation)
64                  return;
65
66              _animation = animation;
67
68              Stop();
69          }
70
71          /// <summary>
72          /// Stop the animation
73          /// </summary>
74          public void Stop()
75          {
76              _timer = 0f;
77
78              _animation.CurrentFrame = 1;
79          }
80
81          /// <summary>
82          /// Draws the current frame
83          /// </summary>
84          /// <param name="spriteBatch">Helper class for drawing strings ↩
                and sprites</param>
85          public void Draw(SpriteBatch spriteBatch)
86          {
87              spriteBatch.Draw(_animation.Texture,
88                               Position,
89                               new Rectangle(_animation.CurrentFrame * ↩
                                     _animation.FrameWidth,
90                                           0,
91                                           _animation.FrameWidth,
92                                           _animation.FrameHeight),
93                               Color.White);
94          }
95      }
96  }
```

Listing 1.11 – ./DonkeyKong/DonkeyKong/Managers/AnimationManager.cs

1.5.2 Input.cs

```
1     /***
2   * Program : DonkeyKong
3   * Author : Tiago Gama
4   * Project : TPI 2020
5   * Date : 25.05.2020 - 09.06.2020
6   * Version : 1.0
7   * Description : Recreation of the original Donkey Kong game by Nintendo
8   ***/
9   using Microsoft.Xna.Framework.Input;
10  using System;
11  using System.Collections.Generic;
12  using System.Linq;
13  using System.Text;
14  using System.Threading.Tasks;
15
16
17  namespace DonkeyKong.Managers
18  {
19      /// <summary>
20      /// This class contains all the potentially necessary inputs for ↩
            the game.
21      /// Not all objects who use this class will have all Inputs set, ↩
            some might not need all of them;
22      /// Inspiration : https://github.com/Oyyou/MonoGame_Tutorials
23      /// </summary>
24      class Input
25      {
26          public Keys Down { get; set; }
```

```
27
28          public Keys Left { get; set; }
29
30          public Keys Right { get; set; }
31
32          public Keys Up { get; set; }
33
34          public List<Keys> Action { get; set; }
35      }
36  }
```

Listing 1.12 – ./DonkeyKong/DonkeyKong/Managers/Input.cs

1.5.3 ScoreManager.cs

```
1     /***
2   * Program : DonkeyKong
3   * Author : Tiago Gama
4   * Project : TPI 2020
5   * Date : 25.05.2020 - 09.06.2020
6   * Version : 1.0
7   * Description : Recreation of the original Donkey Kong game by Nintendo
8   ***/
9   using DonkeyKong.Models;
10  using System;
11  using System.Collections.Generic;
12  using System.IO;
13  using System.Linq;
14  using System.Text;
15  using System.Threading.Tasks;
16  using System.Xml.Serialization;
17
18  namespace DonkeyKong.Managers
19  {
20      /// <summary>
21      /// Manages all the saving and loading highscores process
22      /// Inspiration : ↩
        https://github.com/Oyyou/MonoGame_Tutorials/tree/master/MonoGame_Tutorials/Tu
23      /// </summary>
24      public class ScoreManager
25      {
26          // Since we don't give a path, this'll be saved in the "bin" ↩
            folder
27          private static string _fileName = "scores.xml";
28
29          public List<Score> Highscores { get; private set; }
30
31          public List<Score> Scores { get; private set; }
32
33          public ScoreManager()
34            : this(new List<Score>())
35          {
36
37          }
38
39          public ScoreManager(List<Score> scores)
40          {
41              Scores = scores;
42
43              UpdateHighscores();
44          }
45
46          /// <summary>
47          /// Adds a new score to the list and updates the highscore
48          /// </summary>
49          /// <param name="score">Player's points</param>
50          public void Add(Score score)
51          {
52              Scores.Add(score);
```

```
53
54              // Orders the list so that the lower scores are first ↩
                    ("lower" in terms of number, the lower the score the ↩
                    better)
55              Scores = Scores.OrderBy(c => c.Value).ToList();
56
57              UpdateHighscores();
58          }
59
60          /// <summary>
61          /// Loads the highscore from the file
62          /// </summary>
63          /// <returns></returns>
64          public static ScoreManager Load()
65          {
66              // If there isn't a file to load - create a new instance of ↩
                    "ScoreManager"
67              if (!File.Exists(_fileName))
68                  return new ScoreManager();
69
70              // Otherwise we load the file
71              using (var reader = new StreamReader(new ↩
                    FileStream(_fileName, FileMode.Open)))
72              {
73                  var serializer = new XmlSerializer(typeof(List<Score>));
74
75                  var scores = (List<Score>)serializer.Deserialize(reader);
76
77                  return new ScoreManager(scores);
78              }
79          }
80
81          /// <summary>
82          /// Since the list is ordered, gets the first member of the ↩
                list which is the best score
83          /// </summary>
84          public void UpdateHighscores()
85          {
86              Highscores = Scores.Take(1).ToList(); // Takes the first 1 ↩
                    elements
87          }
88
89          /// <summary>
90          /// Writes the xml file with the highscore
91          /// </summary>
92          /// <param name="scoreManager"></param>
93          public static void Save(ScoreManager scoreManager)
94          {
95              // Overrides the file if it alreadt exists
96              using (var writer = new StreamWriter(new ↩
                    FileStream(_fileName, FileMode.Create)))
97              {
98                  var serializer = new XmlSerializer(typeof(List<Score>));
99
100                 serializer.Serialize(writer, scoreManager.Highscores);
101             }
102         }
103     }
104 }
```

Listing 1.13 – ./DonkeyKong/DonkeyKong/Managers/ScoreManager.cs

1.6 GameComponents

1.6.1 Barrel.cs

```
1   using   DonkeyKong.Models;
2  using DonkeyKong.Sprites;
3  using Microsoft.Xna.Framework;
4  using Microsoft.Xna.Framework.Graphics;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10
11 namespace DonkeyKong.GameComponents
12 {
13     class Barrel : MovingAnimatedSprite
14     {
15
16         GraphicsDevice _graphicsDevice;
17
18         /// <summary>
19         /// Barrel constructor, creates a barrel object that moves ↩
               independantly
20         /// </summary>
21         /// <param name="game">The game variable</param>
22         /// <param name="animations">All barrel animations</param>
23         /// <param name="graphicsDevice">Information about the screen ↩
               used to display the game</param>
24         public Barrel(Game game, Dictionary<string, Animation> ↩
               animations, GraphicsDevice graphicsDevice) : base(game, ↩
               animations)
25         {
26
27             //For the initial barrel position, in the beginning the ↩
                   hitbox will not be set, so we use these
28             _height = _animations.First().Value.FrameHeight;
29             _width = _animations.First().Value.FrameWidth;
30
31             _graphicsDevice = graphicsDevice;
32
33             Speed.X = 3f;
34         }
35
36         /// <summary>
37         ///
38         /// </summary>
39         /// <param name="gameTime"></param>
40         /// <param name="groundLayout"></param>
41         public void Update(GameTime gameTime, Dictionary<string, ↩
               List<Brick>> groundLayout)
42         {
43             base.Update(gameTime);
44             hitbox = new Rectangle((int)_position.X, (int)_position.Y, ↩
                   _width, _height);
45             Move();
46             CollisionBricks(groundLayout);
47         }
48
49         /// <summary>
50         /// A barrel starts the game going right, then everytime it ↩
               hits a wall the speed is inversed,
51         /// </summary>
52         private void Move()
53         {
54             if (hitbox.Right >= _graphicsDevice.Viewport.Width ||
55                 hitbox.Left <= 0)
56             {
57                 Speed.X = -Speed.X;
58             }
59             Velocity.X = Speed.X;
```

```
60              Velocity.Y = Speed.Y;
61
62          }
63
64          /// <summary>
65          /// Stops the barrel from falling down if there is a brick
66          /// </summary>
67          /// <param name="groundLayout">All bricks in the game</param>
68          private void CollisionBricks(Dictionary<string, List<Brick>> ↩
                groundLayout)
69          {
70              foreach (List<Brick> lb in groundLayout.Values)
71              {
72                  foreach (Brick b in lb)
73                  {
74                      if (IsTouchingTop(b))
75                      {
76                          Velocity.Y = 0;
77                      }
78                  }
79              }
80
81          }
82
83          /// <summary>
84          /// Checks if the barrel has reached the oil barrel.
85          /// </summary>
86          /// <param name="oilBarrel"></param>
87          /// <returns></returns>
88          public bool IsCollidingWithOilBarrel(AnimatedSprite oilBarrel)
89          {
90              return this.hitbox.Intersects(oilBarrel.hitbox);
91          }
92
93          /// <summary>
94          /// Checks if the barrel is colliding with only the top part of ↩
                a sprite
95          /// </summary>
96          /// <param name="sprite">A sprite object with an hitbox</param>
97          protected bool IsTouchingTop(GenericSprite sprite)
98          {
99              return this.hitbox.Bottom + this.Velocity.Y > ↩
                    sprite.hitbox.Top &&
100               this.hitbox.Top < sprite.hitbox.Top &&
101               this.hitbox.Right > sprite.hitbox.Left &&
102               this.hitbox.Left < sprite.hitbox.Right;
103         }
104
105     }
106 }
```

Listing 1.14 – ./DonkeyKong/DonkeyKong/GameComponents/Barrel.cs

1.6.2 Brick.cs

```
1   using   DonkeyKong.Sprites;
2  using Microsoft.Xna.Framework;
3  using Microsoft.Xna.Framework.Graphics;
4  using System;
5  using System.Collections.Generic;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace DonkeyKong.GameComponents
11 {
12     class Brick : GenericSprite
13     {
14         /// <summary>
```

```
15          /// A brick is a standart Generic sprite that can be cloned
16          /// </summary>
17          /// <param name="game"></param>
18          /// <param name="graphicsDevice"></param>
19          public Brick(Game game) : base (game)
20          {
21          }
22
23          /// <summary>
24          /// Returns a shallow copy of the brick
25          /// </summary>
26          public object Clone()
27          {
28              return MemberwiseClone();
29          }
30
31
32
33      }
34  }
```

Listing 1.15 – ./DonkeyKong/DonkeyKong/GameComponents/Brick.cs

1.6.3 GameTimer.cs

```
1   using   Microsoft.Xna.Framework;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace DonkeyKong.GameComponents
9  {
10      /// <summary>
11      /// Inspiration pour la classe ↩
            https://www.youtube.com/watch?v=-2FeSrYT1KE
12      /// </summary>
13      class GameTimer : GameComponent
14      {
15
16          private string text;
17          private float time;
18          private bool started;
19          private bool paused;
20          private bool finished;
21
22          public string Text { get => text; set => text = value; }
23          public bool Started { get => started; set => started = value; }
24          public bool Paused { get => paused; set => paused = value; }
25          public bool Finished { get => finished; set => finished = ↩
                value; }
26
27          /// <summary>
28          /// GameTimer constructor, creates a new game timer which ↩
                starts unset and needs to be started
29          /// </summary>
30          /// <param name="game">The game variable</param>
31          /// <param name="startTime">When does the timer start</param>
32          public GameTimer(Game game, float startTime): base(game)
33          {
34              time = startTime;
35              Started = false;
36              Paused = false;
37              Finished = false;
38          }
39
40          /// <summary>
41          /// Runs the timer logic
```

```
42          /// </summary>
43          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
44          public override void Update(GameTime gameTime)
45          {
46
47              float elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;
48
49              if (started && !paused)
50              {
51                  time += elapsed;
52              }
53
54              Text = ((int)time).ToString();
55
56
57              base.Update(gameTime);
58          }
59
60
61      }
62  }
```

Listing 1.16 – ./DonkeyKong/DonkeyKong/GameComponents/GameTimer.cs

1.6.4 Kong.cs

```
1   using   DonkeyKong.Models;
2   using DonkeyKong.Sprites;
3   using Microsoft.Xna.Framework;
4   using Microsoft.Xna.Framework.Audio;
5   using Microsoft.Xna.Framework.Graphics;
6   using Microsoft.Xna.Framework.Input;
7   using System;
8   using System.Collections.Generic;
9   using System.Linq;
10  using System.Text;
11  using System.Threading.Tasks;
12
13  namespace DonkeyKong.GameComponents
14  {
15      class Kong : AnimatedSprite
16      {
17          private float timer;
18          private Random rnd = new Random();
19          private bool _firstTimeOnAnimation;
20
21          private bool isDoneWithThrowingBarrel = false;
22
23
24          Dictionary<string, SoundEffect> _soundEffects;
25          SoundEffectInstance _soundInstance;
26
27          /// <summary>
28          /// Kong constructor with animations and sound effects
29          /// </summary>
30          /// <param name="game">The game variable</param>
31          /// <param name="animations">All kong animations</param>
32          /// <param name="soundEffects">All kong sound effects</param>
33          public Kong(Game game, Dictionary<string, Animation> ↩
                animations, Dictionary<string, SoundEffect> soundEffects) : ↩
                base(game, animations)
34          {
35              //Change the default frame speed
36              foreach (Animation a in animations.Values)
37              {
38                  a.FrameSpeed = 0.5f;
39              }
40
```

```
41              //Kong sound effects
42              _soundEffects = soundEffects;
43              _soundInstance = _soundEffects["Idle"].CreateInstance();
44
45              //Initializing kong variables at the start of the game
46              _firstTimeOnAnimation = false;
47              timer = 1;
48              rnd = new Random();
49          }
50
51          /// <summary>
52          /// Every time the timer reaches 0 start the grabBarrel animation
53          /// </summary>
54          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
55          protected void SetAnimations(GameTime gameTime)
56          {
57
58              float elapsed = (float)gameTime.ElapsedGameTime.TotalSeconds;
59              timer -= elapsed;
60
61              //Timer expired, execute action
62              if (timer < 0)
63              {
64                  //Reset Timer between 2s and 4s
65                  timer = rnd.Next(2, 5);
66                  _animationManager.Play(_animations["GrabBarrel"]);
67
68              }
69          }
70
71          /// <summary>
72          /// Runs the kong logic
73          /// </summary>
74          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
75          public override void Update(GameTime gameTime)
76          {
77              SetAnimations(gameTime);
78              AnimationLogic();
79
80              base.Update(gameTime);
81          }
82
83          /// <summary>
84          /// Checks if its the first time on an animation
85          /// </summary>
86          private void AnimationLogic()
87          {
88              if (_animationManager._animation == ↩
                    _animations["GrabBarrel"] && ↩
                    _animations["GrabBarrel"].CurrentFrame == 1)
89              {
90                  _firstTimeOnAnimation = true;
91              }
92              else if (_animationManager._animation == ↩
                    _animations["GrabBarrel"] && ↩
                    _animations["GrabBarrel"].CurrentFrame == 2)
93              {
94                  _firstTimeOnAnimation = false;
95              }
96
97              if (_animationManager._animation == ↩
                    _animations["GrabBarrel"] && _soundInstance.State != ↩
                    SoundState.Playing)
98              {
99                  _soundInstance.Play();
100             }
101         }
102
103         /// <summary>
```

```
104         /// If its the first time on the grabBarrel animation return ↩
                that you can now spawn a barrel.
105         /// If its not than start the idle animation
106         /// </summary>
107         public bool CanSpawnBarrel()
108         {
109             bool canSpawnBarrel = false;
110             if (_animationManager._animation == ↩
                    _animations["GrabBarrel"] && ↩
                    _animations["GrabBarrel"].CurrentFrame == 2 && ↩
                    _firstTimeOnAnimation == true)
111             {
112                 canSpawnBarrel = true;
113                 isDoneWithThrowingBarrel = canSpawnBarrel;
114 
115             }
116 
117 
118             if (isDoneWithThrowingBarrel == true && ↩
                    _animationManager._animation == ↩
                    _animations["GrabBarrel"] && ↩
                    _animations["GrabBarrel"].CurrentFrame == 0)
119             {
120                 isDoneWithThrowingBarrel = false;
121                 _animationManager.Play(_animations["Idle"]);
122 
123 
124             }
125             return canSpawnBarrel;
126         }
127 
128 
129 
130 
131     }
132 }
```

Listing 1.17 – ./DonkeyKong/DonkeyKong/GameComponents/Kong.cs

1.6.5 Ladder.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using DonkeyKong.Sprites;
10 using Microsoft.Xna.Framework;
11 using Microsoft.Xna.Framework.Graphics;
12 
13 namespace DonkeyKong.GameComponents
14 {
15     /// <summary>
16     /// A ladder which will be used to trigger mario's climbing movement
17     /// </summary>
18     class Ladder : GenericSprite
19     {
20         int _nbSpritesInStack;
21 
22         public int NbSpritesInStack { get => _nbSpritesInStack; set => ↩
                _nbSpritesInStack = value; }
23 
24 
25         /// <param name="game">The game variable</param>
26         public Ladder(Game game):base(game)
27         {
```

```
28
29          }
30
31          /// <summary>
32          /// Runs the ladder logic, where the collisions lies etc
33          /// </summary>
34          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
35          public override void Update(GameTime gameTime)
36          {
37              int collisionHeight = _texture.Height * NbSpritesInStack;
38
39              hitbox = new Rectangle(
40                      (int)_position.X,
41                      (int)_position.Y,
42                      _texture.Width,
43                      collisionHeight);
44          }
45
46          /// <summary>
47          /// Draws the entire ladder, which is composed of mini ladder ↩
                parts
48          /// </summary>
49          /// <param name="spriteBatch">Helper class for drawing strings ↩
                and sprites</param>
50          public override void Draw(SpriteBatch spriteBatch)
51          {
52              for (int i = 0; i < NbSpritesInStack; i++)
53              {
54                  spriteBatch.Draw(_texture, new ↩
                        Rectangle((int)_position.X, (int)_position.Y + ↩
                        _texture.Height * i, _texture.Width, ↩
                        _texture.Height), Color.White);
55              }
56
57          }
58
59      }
60  }
```

Listing 1.18 – ./DonkeyKong/DonkeyKong/GameComponents/Ladder.cs

1.6.6 Mario.cs

```
1     /***
2   * Program : DonkeyKong
3   * Author : Tiago Gama
4   * Project : TPI 2020
5   * Date : 25.05.2020 - 09.06.2020
6   * Version : 1.0
7   * Description : Recreation of the original Donkey Kong game by Nintendo
8   ***/
9   using DonkeyKong.Models;
10  using DonkeyKong.Sprites;
11  using Microsoft.Xna.Framework;
12  using Microsoft.Xna.Framework.Audio;
13  using Microsoft.Xna.Framework.Graphics;
14  using Microsoft.Xna.Framework.Input;
15  using System.Collections.Generic;
16  using System.Linq;
17
18  namespace DonkeyKong.GameComponents
19  {
20      /// <summary>
21      /// A Mario which can be moved with the keys given to the Input ↩
            property.
22      /// Mario has different sets of animations and sounds.
23      /// He checks for collisions with ladders bricks and barrels and ↩
            behaves differently when colliding with each of these objects
```

```
24      /// </summary>
25      class Mario : MovingAnimatedSprite
26      {
27          #region Variables and properties
28          private Rectangle _hitbox;
29          private GraphicsDevice _graphicsDevice;
30
31          private bool _inGame;
32          private bool _isGoingRight;
33          private bool _marioCollidedWithBarrel;
34
35          private bool _onGround;
36          private bool _hasJumped;
37          private bool _onLaddder;
38
39          private Dictionary<string, SoundEffect> _soundEffects;
40          private SoundEffectInstance _soundInstanceJump;
41          private SoundEffectInstance _soundInstanceWalking;
42          private SoundEffectInstance _soundInstanceClimbing;
43
44
45          public Rectangle Hitbox { get => _hitbox; set => _hitbox = ↩
                value; }
46
47          #endregion
48
49
50          /// <param name="game">The game variable</param>
51          /// <param name="animations">All mario animations</param>
52          /// <param name="soundEffects">All mario sound effects</param>
53          /// <param name="graphicsDevice">Information about the screen ↩
                used to display the game</param>
54          /// <param name="inGame">Is mario in the game</param>
55          public Mario(Game game, Dictionary<string, Animation> ↩
                animations, Dictionary<string, SoundEffect> soundEffects, ↩
                GraphicsDevice graphicsDevice, bool inGame = false) : ↩
                base(game, animations)
56          {
57              _inGame = inGame;
58              _graphicsDevice = graphicsDevice;
59
60              //For the initial mario position, in the beginning the ↩
                    hitbox will not be set, so we use these
61              _height = _animations.First().Value.FrameHeight;
62              _width = _animations.First().Value.FrameWidth;
63
64              //All mario sound effects
65              _soundEffects = soundEffects;
66              _soundInstanceWalking = ↩
                    _soundEffects["Walking"].CreateInstance();
67              _soundInstanceWalking.Volume = 0.1f;
68              _soundInstanceJump = _soundEffects["Jump"].CreateInstance();
69              _soundInstanceJump.Volume = 0.5f;
70              _soundInstanceClimbing = ↩
                    _soundEffects["Climbing"].CreateInstance();
71
72              //Initializw mario variables at the start of the game
73              _isGoingRight = true;
74              _hasJumped = true;
75              _onGround = true;
76              _onLaddder = false;
77              _marioCollidedWithBarrel = true;
78
79          }
80
81          #region Updates
82          /// <summary>
83          /// Runs the Mario logic he is not in the game.
84          /// </summary>
85          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
86          public override void Update(GameTime gameTime)
```

```
87          {
88              Hitbox = new Rectangle((int)_position.X, (int)_position.Y, ↩
                    _animations.First().Value.FrameWidth, ↩
                    _animations.First().Value.FrameHeight);
89              Move();
90              ScreenCollisions();
91              SetAnimationsAndSounds();
92              base.Update(gameTime);
93          }
94
95          /// <summary>
96          /// Runs the game logic for Mario, checks for collisions, ↩
                gathers inputs, sets the animations and play audio.
97          /// </summary>
98          /// <param name="gameTime">Provides a snapshot of timing ↩
                values</param>
99          /// <param name="groundLayout">All bricks in the game, used to ↩
                know when mario can walk</param>
100         /// <param name="allLadders">All ladders in the game, used to ↩
                know when mario can climb</param>
101         /// <param name="allBarrels">All barrels in the game, used to ↩
                know when mario dies</param>
102         public void Update(GameTime gameTime, Dictionary<string, ↩
                List<Brick>> groundLayout, List<Ladder> allLadders, ↩
                List<Barrel> allBarrels)
103         {
104             //Updates Mario's hitbox
105             Hitbox = new Rectangle((int)_position.X, (int)_position.Y, ↩
                    _animations.First().Value.FrameWidth, ↩
                    _animations.First().Value.FrameHeight);
106
107             //Updates mario's sounds and animations
108             SetAnimationsAndSounds();
109             _animationManager.Update(gameTime);
110
111             //Makes sure Mario doesn't go off screen
112             ScreenCollisions();
113
114             //Checks Mario's collision with barrels and ladders
115             BarrelCollision(allBarrels);
116             LadderCollision(allLadders, groundLayout);
117
118             //If Mario is not on a ladder
119             if (!_onLaddder)
120             {
121                 //He is able to move normally
122                 Move();
123                 GroundCollision(groundLayout);
124                 ApplyPhysics(gameTime);
125             }
126             else
127             {
128                 //He can only move upwards or downwards until he is no ↩
                        longer in a ladder
129                 LadderMovement();
130             }
131
132
133             //Updates Mario's position with his current velocity
134             Position += Velocity;
135
136         }
137
138         #endregion
139
140         #region Methods
141
142         #region Movement
143         /// <summary>
144         /// Mario's walking and jumping are controlled through here.
145         /// </summary>
146         private void Move()
```

```
147         {
148             //If he is not in the game. He is in one of the menus
149             if (!_inGame)
150             {
151                 //Allow him to go up and down freely
152                 if (Keyboard.GetState().IsKeyDown(Input.Up))
153                     Velocity.Y = -Speed.Y;
154                 if (Keyboard.GetState().IsKeyDown(Input.Down))
155                     Velocity.Y = Speed.Y;
156             }
157             else//If he is in the game he will be able to jump
158             {
159                 Jump();
160             }
161 
162             //Sets the Mario velocity considering which input was given
163             if (Keyboard.GetState().IsKeyDown(Input.Left))
164             {
165                 Velocity.X = -Speed.X;
166                 _isGoingRight = false;
167             }
168             else if (Keyboard.GetState().IsKeyDown(Input.Right))
169             {
170                 Velocity.X = Speed.X;
171                 _isGoingRight = true;
172             }
173             else
174                 Velocity.X = 0;
175 
176 
177         }
178 
179         /// <summary>
180         /// If Mario is on the ground he will be able to jump
181         /// </summary>
182         private void Jump()
183         {
184 
185             if (_onGround)
186             {
187                 foreach (Keys k in Input.Action)
188                 {
189                     if (Keyboard.GetState().IsKeyDown(k) && _hasJumped ↩
                           == false)
190                     {
191                         _position.Y -= 5f;
192                         Velocity.Y = -4f;
193                         _hasJumped = true;
194                     }
195                 }
196             }
197 
198 
199         }
200 
201         /// <summary>
202         /// If Mario is on a ladder he can move upwards or downwards ↩
              but cannot move sideways
203         /// </summary>
204         private void LadderMovement()
205         {
206             if (Keyboard.GetState().IsKeyDown(Input.Up))
207                 Velocity.Y = -Speed.Y;
208             else if (Keyboard.GetState().IsKeyDown(Input.Down))
209                 Velocity.Y = Speed.Y;
210             else
211                 Velocity.Y = 0;
212 
213             Velocity.X = 0;
214         }
215 
216         /// <summary>
```

```
217         /// Adds gravity to mario
218         /// </summary>
219         /// <param name="gameTime">Provides a snapshot of timing values</param>
220         private void ApplyPhysics(GameTime gameTime)
221         {
222 
223             if (_hasJumped == true || _onGround == false)
224             {
225                 float i = 1;
226                 Velocity.Y += 0.15f * i;
227             }
228 
229             if (_onGround == true)
230             {
231                 Velocity.Y = 0;
232                 _hasJumped = false;
233             }
234 
235         }
236 
237         #endregion
238 
239         #region All collisions
240 
241         #region Game objects collisions
242         /// <summary>
243         /// Checks if Mario's collision with the ground, and since the ground is tilted moves mario upwards ito compensate it
244         /// </summary>
245         /// <param name="groundLayout"></param>
246         private void GroundCollision(Dictionary<string, List<Brick>> groundLayout)
247         {
248             _onGround = false;
249 
250             foreach (List<Brick> lb in groundLayout.Values)
251             {
252 
253                 for (int i = 0; i < lb.Count; i++)
254                 {
255                     //If standing on a brick
256                     if (this.IsTouchingTop(lb[i]) && !IsTouchingBottom(lb[i]))
257                     {
258 
259                         if (i > 0 && _hasJumped == false)
260                         {
261                             //If going left or right and colliding with one of the bricks sides
262                             if (this.IsTouchingRight(lb[i - 1]) ||
263                             this.IsTouchingLeft(lb[i - 1]))
264                             {
265                                 _position.Y = lb[i - 1]._position.Y - _height;
266                             }
267                         }
268 
269                         //Set the mario game variables
270                         _hasJumped = false;
271                         _onGround = true;
272                         _onLaddder = false;
273 
274                     }
275                 }
276 
277             }
278         }
279 
280         /// <summary>
281         /// Checks if Mario's hitbox collided with any of the barrels hitboxes
```

```
        /// </summary>
        /// <param name="allBarrels">All barrels in the game</param>
        private void BarrelCollision(List<Barrel> allBarrels)
        {
            _marioCollidedWithBarrel = false;
            foreach (Barrel b in allBarrels)
            {
                if (Hitbox.Intersects(b.hitbox))
                {
                    _marioCollidedWithBarrel = true;

                    StopAllSoundInstances();
                }

            }
        }

        /// <summary>
        /// Checks if Mario's hitbox collided with any of the ladders hitboxes
        /// </summary>
        /// <param name="allLadders">All ladders in the game</param>
        /// <param name="groundLayout">All bricks in the game</param>
        private void LadderCollision(List<Ladder> allLadders, Dictionary<string, List<Brick>> groundLayout)
        {
            _onLaddder = false;

            foreach (Ladder l in allLadders)
            {
                if (Hitbox.Intersects(l.hitbox))
                {
                    //Mario has to be in the center of the ladder to be able to use it
                    if (((IsTouchingRight(l) && Position.X + _width / 2 <= l._position.X + l._texture.Width / 2)
                        || (IsTouchingLeft(l) && Position.X + _width / 2 >= l._position.X + l._texture.Width / 2))
                        && (Keyboard.GetState().IsKeyDown(Input.Up) || Keyboard.GetState().IsKeyDown(Input.Down)))
                    {
                        _onLaddder = true;

                        //If he is going downwards make sure he isnt in the ground
                        if (Keyboard.GetState().IsKeyDown(Input.Down))
                        {
                            GroundCollision(groundLayout);
                        }
                    }

                }
            }
        }


        #endregion

        /// <summary>
        /// Stop Mario from going out of the screen
        /// </summary>
        private void ScreenCollisions()
        {
            if (Hitbox.Right >= _graphicsDevice.Viewport.Width)
                _position.X = _graphicsDevice.Viewport.Width - _width;
            if (Hitbox.Left <= 0)
                _position.X = 0;
            if (Hitbox.Top <= 0)
                _position.Y = 0;
            if (Hitbox.Bottom >= _graphicsDevice.Viewport.Height)
                _position.Y = _graphicsDevice.Viewport.Height - _height;
        }
```

```
347
348         #region Collision checks
349
350         /// <summary>
351         /// Checks if Mario is going to collide with the left of a sprite and he is not past the sprite's left
352         /// </summary>
353         /// <param name="sprite">A sprite object with an hitbox</param>
354         private bool IsTouchingLeft(GenericSprite sprite)
355         {
356             return this.Hitbox.Right + this.Velocity.X > sprite.hitbox.Left &&
357               this.Hitbox.Left < sprite.hitbox.Left;
358         }
359
360         /// <summary>
361         /// Checks if Mario is going to collide with the right of an sprite and he is not past the sprite's right
362         /// </summary>
363         /// <param name="sprite">A sprite object with an hitbox</param>
364         private bool IsTouchingRight(GenericSprite sprite)
365         {
366             return this.Hitbox.Left + this.Velocity.X <= sprite.hitbox.Right &&
367               this.Hitbox.Right > sprite.hitbox.Right;
368         }
369
370         /// <summary>
371         /// Checks if Mario is colliding with only the top part of a sprite
372         /// </summary>
373         /// <param name="sprite">A sprite object with an hitbox</param>
374         private bool IsTouchingTop(GenericSprite sprite)
375         {
376             return this.Hitbox.Bottom + this.Velocity.Y >= sprite.hitbox.Top &&
377               this.Hitbox.Top < sprite.hitbox.Top &&
378               this.Hitbox.Right > sprite.hitbox.Left &&
379               this.Hitbox.Left < sprite.hitbox.Right;
380         }
381
382         /// <summary>
383         /// Checks if Mario is colliding with only the bottom part of a sprite
384         /// </summary>
385         /// <param name="sprite">A sprite object with an hitbox</param>
386         private bool IsTouchingBottom(GenericSprite sprite)
387         {
388             return this.Hitbox.Top + this.Velocity.Y < sprite.hitbox.Bottom &&
389               this.Hitbox.Bottom > sprite.hitbox.Bottom &&
390               this.Hitbox.Right > sprite.hitbox.Left &&
391               this.Hitbox.Left < sprite.hitbox.Right;
392         }
393         #endregion
394
395         #endregion
396         /// <summary>
397         /// Returns if mario collided with one of the barrels
398         /// </summary>
399         public bool IsMarioDead()
400         {
401             return _marioCollidedWithBarrel;
402         }
403
404         #region Mario's animations and sounds
405         /// <summary>
406         /// All mario animations and sounds are controlled through here.
407         /// Jumping, walking, climbing ladders or being idle.
408         /// </summary>
409         protected override void SetAnimationsAndSounds()
410         {
```

```
411             //If mario is moving upwords or downwards in ladder, he is climbing
412             if ((Velocity.Y < 0 || Velocity.Y > 0) && _inGame && _onLaddder)
413             {
414                 //Play the climbing animation
415                 _animationManager.Play(_animations["Climb"]);
416                 //If the climbing sound is not already playing, play it
417                 if (_soundInstanceClimbing.State != SoundState.Playing)
418                 {
419                     _soundInstanceClimbing.Play();
420                 }
421                 //If the walking sound is playing stop it
422                 if (_soundInstanceWalking.State == SoundState.Playing)
423                 {
424                     _soundInstanceWalking.Stop();
425                 }
426                 //If the jumping sound is playing stop it
427                 if (_soundInstanceJump.State == SoundState.Playing)
428                 {
429                     _soundInstanceJump.Stop();
430                 }
431 
432 
433             }//If mario is Jumping
434             else if (Velocity.Y < 0 && _inGame)
435             {
436                 //Check if he last was going right or left to start the correct animation
437                 if (_isGoingRight)
438                 {
439                     _animationManager.Play(_animations["JumpRight"]);
440                 }
441                 else
442                 {
443                     _animationManager.Play(_animations["JumpLeft"]);
444                 }
445 
446                 //If the walking sound is playing stop it
447                 if (_soundInstanceWalking.State == SoundState.Playing)
448                 {
449                     _soundInstanceWalking.Stop();
450                 }
451                 //If the jumping sound is not already playing, play it
452                 if (_soundInstanceJump.State != SoundState.Playing)
453                 {
454                     _soundInstanceJump.Play();
455                 }
456 
457             }//If going right
458             else if (Velocity.X > 0 && _onGround)
459             {
460                 //Start the walking right animation
461                 _animationManager.Play(_animations["WalkRight"]);
462 
463                 //If the walking sound is not already playing, play it
464                 if (_soundInstanceWalking.State != SoundState.Playing && _onGround)
465                 {
466                     _soundInstanceWalking.Play();
467                 }
468 
469             }//If going left
470             else if (Velocity.X < 0 && _onGround)
471             {
472                 //Start the walking left animation
473                 _animationManager.Play(_animations["WalkLeft"]);
474 
475                 //If the walking sound is not already playing, play it
476                 if (_soundInstanceWalking.State != SoundState.Playing && _onGround)
477                 {
```

```
478                     _soundInstanceWalking.Play();
479                 }
480 
481             }//If movement is stopped, aka mario is idle
482             else if (Velocity.X == 0 && _onGround)
483             {
484                 //If he last was going right
485                 if (_isGoingRight)
486                 {
487                     _animationManager.Play(_animations["WalkRight"]);
488                 }
489                 else
490                 {
491                     _animationManager.Play(_animations["WalkLeft"]);
492                 }
493 
494                 //Stop the animation at the second frame
495                 _animationManager.Stop();
496 
497                 //If the walking sound is playing stop it
498                 if (_soundInstanceWalking.State == SoundState.Playing)
499                 {
500                     _soundInstanceWalking.Stop();
501                 }
502             }
503             else // In case mario is not in one of the other ↩
                    conditions, but should not occur
504             {
505                 _animationManager.Stop();
506             }
507 
508 
509 
510 
511         }
512 
513 
514         /// <summary>
515         /// Used to reset sound instances.
516         /// For example when Mario dies we want the sounds to stop so ↩
                they dont carry over to his next life.
517         /// </summary>
518         public void StopAllSoundInstances()
519         {
520             _soundInstanceJump.Stop();
521             _soundInstanceWalking.Stop();
522             _soundInstanceClimbing.Stop();
523         }
524         #endregion
525 
526         #endregion
527 
528     }
529 }
```

Listing 1.19 – ./DonkeyKong/DonkeyKong/GameComponents/Mario.cs

1.7 Controls

1.7.1 MenuButton.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Recreation of the original Donkey Kong game by Nintendo
8 ***/
9 using DonkeyKong.Managers;
10 using DonkeyKong.Sprites;
11 using Microsoft.Xna.Framework;
12 using Microsoft.Xna.Framework.Graphics;
13 using Microsoft.Xna.Framework.Input;
14
15 namespace DonkeyKong.Controls
16 {
17
18     /// <summary>
19     /// A sprite that changes color when colliding with an object
20     /// </summary>
21     class MenuButton : GenericSprite
22     {
23         bool collisionWithMario;
24
25         public Input Input;
26
27         public MenuButton(Game game) : base(game)
28         {
29             collisionWithMario = false;
30         }
31
32
33         /// <summary>
34         /// Runs the button logic.
35         /// </summary>
36         /// <param name="gameTime">Provides a snapshot of timing ↩
               values</param>
37         /// <param name="collider">An object that can collide with the ↩
               button</param>
38         public void Update(GameTime gameTime, Rectangle collider)
39         {
40
41             base.Update(gameTime);
42             CollisionWithCollider(collider);
43
44         }
45
46         /// <summary>
47         /// Checks if the collider (in this case Mario) is colliding ↩
               with the button.
48         /// </summary>
49         /// <param name="collider">An object that can collide with the ↩
               button</param>
50         private void CollisionWithCollider(Rectangle collider)
51         {
52             if (collider.Intersects(hitbox))
53             {
54                 collisionWithMario = true;
55             }
56             else
57             {
58                 collisionWithMario = false;
59             }
60         }
61
62         /// <summary>
```

```
63          /// Returns if any of the action inputs were pressed while ↩
                mario is on top of the button.
64          /// </summary>
65          public bool ButtonPressed()
66          {
67              bool pressed = false;
68              KeyboardState keyboardState = Keyboard.GetState();
69
70              foreach (Keys k in Input.Action)
71              {
72                  if (collisionWithMario && keyboardState.IsKeyDown(k))
73                  {
74                      pressed = true;
75                  }
76              }
77              return pressed;
78          }
79
80          /// <summary>
81          /// Draws the button with the color white or if its colliding ↩
                with something draw it with the color grey to represent it.
82          /// </summary>
83          /// <param name="spriteBatch">Helper class for drawing strings ↩
                and sprites</param>
84          public override void Draw(SpriteBatch spriteBatch)
85          {
86              //Draw the sprite
87              if (collisionWithMario)
88              {
89                  spriteBatch.Draw(_texture, hitbox, Color.Gray);
90              }
91              else
92              {
93                  spriteBatch.Draw(_texture, hitbox, Color.White);
94              }
95
96          }
97
98
99      }
100 }
```

Listing 1.20 – ./DonkeyKong/DonkeyKong/Controls/MenuButton.cs

1.8 Tests

1.8.1 ScoreManagerTests.cs

```
1    /***
2 * Program : DonkeyKong
3 * Author : Tiago Gama
4 * Project : TPI 2020
5 * Date : 25.05.2020 - 09.06.2020
6 * Version : 1.0
7 * Description : Unit tests for the Donkey Kong program
8 ***/
9 using Microsoft.VisualStudio.TestTools.UnitTesting;
10 using DonkeyKong.Models;
11
12 namespace DonkeyKong.Managers.Tests
13 {
14     [TestClass()]
15     public class ScoreManagerTests
16     {
17         /// <summary>
18         /// Make sure the adding method actually adds a score to the ↩
               score list
19         /// </summary>
20         [TestMethod()]
21         public void AddTest()
22         {
23             ScoreManager scoreManager = new ScoreManager();
24             Score s = new Score()
25             {
26                 PlayerName = "NONAME",
27                 Value = "0025",
28             };
29
30             scoreManager.Add(s);
31
32             CollectionAssert.Contains(scoreManager.Scores,s);
33         }
34
35         /// <summary>
36         /// Make sure you can save and load scores
37         /// </summary>
38         [TestMethod()]
39         public void SaveAndLoadTest()
40         {
41             ScoreManager scoreManager = new ScoreManager();
42             Score s = new Score()
43             {
44                 PlayerName = "NONAME",
45                 Value = "0014",
46             };
47
48             scoreManager.Add(s);
49
50             ScoreManager.Save(scoreManager);
51             scoreManager = ScoreManager.Load();
52
53             Assert.AreEqual(s.Value, scoreManager.Scores[0].Value);
54
55         }
56
57         /// <summary>
58         /// Make sure the highscore is updated
59         /// </summary>
60         [TestMethod()]
61         public void UpdateHighscoresTest()
62         {
63             ScoreManager scoreManager = new ScoreManager();
64             Score s = new Score()
65             {
66                 PlayerName = "NONAME",
```

```
67                  Value = "0025",
68              };
69              scoreManager.Add(s);
70
71              Score currentHighscore = scoreManager.Highscores[0];
72
73              s = new Score()
74              {
75                  PlayerName = "NONAME",
76                  Value = "0014",
77              };
78              scoreManager.Add(s);
79
80              Score newHighscore = scoreManager.Highscores[0];
81
82              //The UpdateHighscores method is called when adding a new ↩
                    score, so no need to call it again
83
84              Assert.AreNotEqual(currentHighscore.Value, ↩
                    newHighscore.Value);
85          }
86
87
88      }
89  }
```

Listing 1.21 – ./DonkeyKong/DonkeyKongTests/Managers/ScoreManagerTests.cs