Software Design Document

Technical difficulties

2016

Contents

[1. Introduction 2](#_Toc450134469)

[1.1 Purpose 2](#_Toc450134470)

[2. Design Considerations 2](#_Toc450134471)

[2.1 Assumptions 2](#_Toc450134472)

[2.2 General Constraints 2](#_Toc450134473)

[2.3 Goals 2](#_Toc450134474)

[3. System Architecture Overview 3](#_Toc450134475)

[3.1 Changes since Production 4](#_Toc450134476)

[3.2 System Interfaces 4](#_Toc450134477)

[3.2.1 IGameObject 4](#_Toc450134478)

[3.2.2 IEntity 4](#_Toc450134479)

[3.2.3 ICollisionListener 4](#_Toc450134480)

[3.2.4 IEntityComponent 4](#_Toc450134481)

[4. Entity Management 5](#_Toc450134482)

[4.1 EntityManager 5](#_Toc450134483)

[4.2 Entity 6](#_Toc450134484)

[EntityComponent Management 8](#_Toc450134485)

[EntityComponent 8](#_Toc450134486)

[5. EntityComponents 9](#_Toc450134487)

[5.1 Transform 9](#_Toc450134488)

[5.2 Text 10](#_Toc450134489)

[5.3 State Driven Behaviours 11](#_Toc450134490)

[5.3.1 StateMachine 11](#_Toc450134491)

[5.3.2 StateTransition 11](#_Toc450134492)

[5.4 Collision Detection 12](#_Toc450134493)

[5.4.1 Collider 12](#_Toc450134494)

[5.4.2 BoxCollider 12](#_Toc450134495)

[5.4.3 CircleCollider 13](#_Toc450134496)

[5.4.4 PlaneCollider 13](#_Toc450134497)

[5.5 Physics 14](#_Toc450134498)

[5.5.1 PhysicsBody 14](#_Toc450134499)

[6. Scene Management 15](#_Toc450134500)

[6.1 SceneManager 15](#_Toc450134501)

[6.2 Scene 16](#_Toc450134502)

[7. Game 17](#_Toc450134503)

[7.1 Floor and Upper Floor 17](#_Toc450134504)

[7.2 Patient 18](#_Toc450134505)

[7.3 Scoreboard 19](#_Toc450134506)

[7.4 ScoreManager 19](#_Toc450134507)

[7.5 ScoreData 20](#_Toc450134508)

[7.6 ScoreDisplay 21](#_Toc450134509)

[7.7 Score 21](#_Toc450134510)

# 1. Introduction

This Software Design Document (or SDD) is a document to provide documentation which will be used to guide the development of the engine by providing the details for how the engine and architecture should be built. Within this SDD is technical and graphical documentation of the software design for the project including class diagrams and data dictionaries.

## 1.1 Purpose

The purpose of this document is to provide a description of the design of a system fully enough to allow for all team members to proceed with an understanding of what is to be built and how it has been conceptualised to be built.

# 2. Design Considerations

This section details any considerations, which need to be addressed before attempting to devise or communicate a complete design solution.

## 2.1 Assumptions

This document is assuming that the user is running the system either as a standalone executable, or as a Visual Studio 2013/2015 project, with MonoGame 3.4 installed. The software depends on the correct installation of MonoGame in order to be run when opening the solution in Visual Studio. This should not affect users who are running the executable.

## 2.2 General Constraints

This system is implementing and using MonoGame version 3.4.0.456. There are no guarantees that this system will work if using a different version of MonoGame/XNA than specified. The system has been worked on in both Visual Studio 2013 and Visual Studio 2015. This system has been written in C#, and as such only has native support for Windows, or other Microsoft operating systems.

## 2.3 Goals

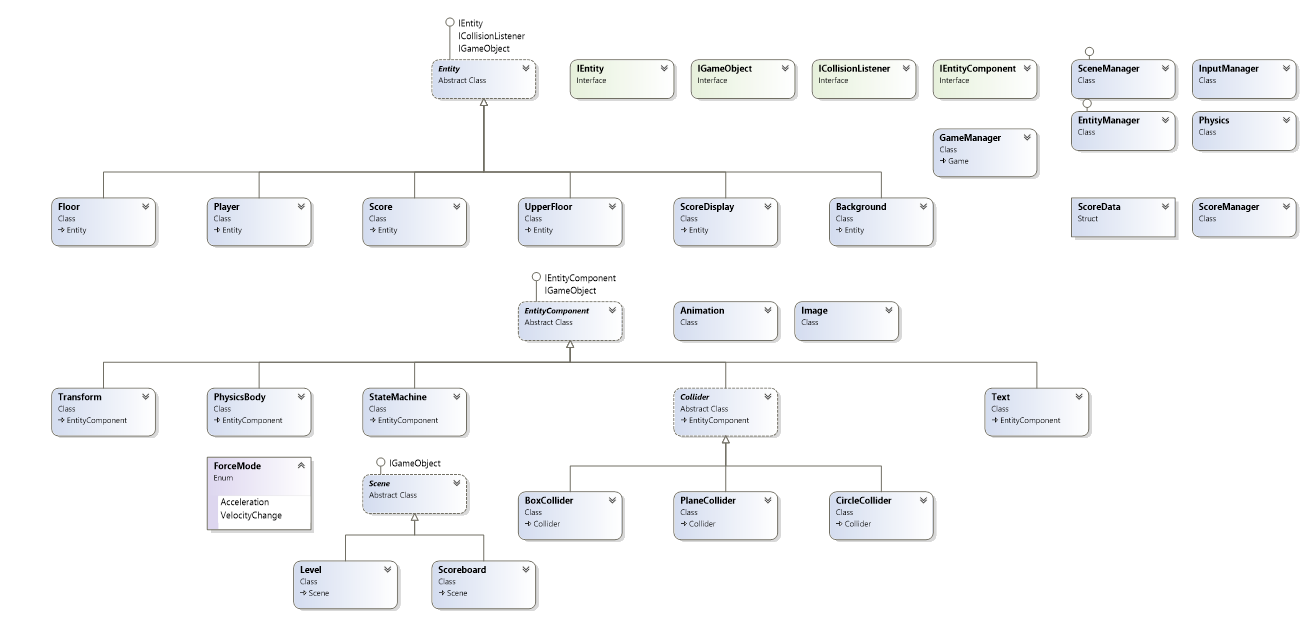
The goal of the design of this software is to produce a back-end system which can be used, alongside MonoGame/XNA to produce videogame software. This document details both the Engine itself, and an example of software built using the engine.

Design Goals for this system include:

* KISS Principle, for ease of use by future software developers
* S.O.L.I.D., for general programming best practices.

# 3. System Architecture Overview

The architecture is Component based. Entities within the system are composed of a list of components, which give them access to generic behaviours based on the components they are implementing. The GameManager Initializes the SceneManager, which contains a list of all scenes in the build. Content for these scenes is loaded when the game is run, and only the current scene is updated or drawn. This means there are no loading times, other than when the game is initially run. Scenes store their own EntityManagers, which have the list of entities for that scene, and handles the Initialization, Loading, Drawing and Updating of these entities.



## 3.1 Changes since Production

For the production milestone, there was an interface (IEntity) but it wasn’t actually used at any point, everything was done through the Entity class. Whilst this worked, it wasn’t what would be considered elegant. It made the system open for both modification and expansion, which is not an ideal scenario. For potential developers using the engine, it is much more efficient to use a series of interfaces that let them know what method they need to implement, than to try and work out how a class works and implements various methods and members.

A number of System Interfaces have since been implemented to handle every object in the engine. These interfaces are detailed fully below.

## 3.2 System Interfaces

### 3.2.1 IGameObject

Most of the objects within this system implement the IGameObject interface, which contains the definition for Initialize, LoadContent, UnloadContent, Draw and Update. These methods are all used very frequently, so an interface is used to ensure that all objects that might need to use one of these methods, is implementing it.

### 3.2.2 IEntity

All Entities implement the IEntity interface. IEntity contains definitions for AddComponent and GetComponent methods, as well as a tag and transform property.

### 3.2.3 ICollisionListener

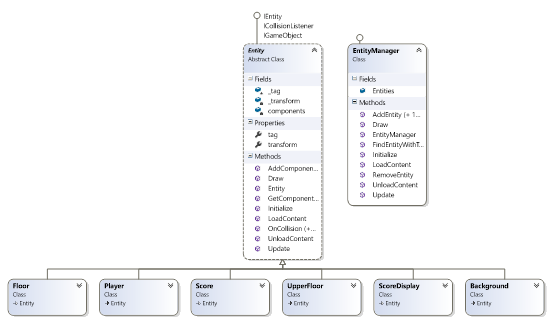
All Entities implement the ICollisionListener interface. This interface contains definition for an OnCollision method, with 2 overloads. Implementation of this interface allows Entities to react to collisions.

### 3.2.4 IEntityComponent

All EntityComponents implement the IEntityComponent interface. Entities store a list of their components, with references to specific components being obtained using the GetComponent method, and casting the type. IEntityComponent contains definitions for an entity property, and a SetEntity method.

# 4. Entity Management

As mentioned above, the architecture is Component based. This means that Entity Management has two layers, management of Entities themselves, and management of their components.



### 4.1 EntityManager

The EntityManager is responsible for storing a list of Entities within a scene, and contains the behaviour for the creation, initialization, loading, drawing and updating of Entities.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| Entities | List< IEntity> | EntityManager stores a list of Entities, as type IEntity. This provides access to the AddComponent and GetComponent methods, as well as a reference to the Transform or tag of the Entity. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| AddEntity(IEntity entity) | void | Adds the entity parameter to the entity list, as type IEntity. Use this to add an already existing entity to the list. |
| AddEntity<T>() where T: IEntity, new() | void | Generic overload of AddEntity(IEntity entity). Adds a new entity of type T to the entity list. T must implement the interface IEntity. Use this method to create a new instance of an entity. |
| Draw(SpriteBatch spriteBatch) | void | Implemented as part of the IGameObject interface. Calls Draw on all entities in the IEntity list, casting them as type IGameObject. |
| FindEntityWithTag(string \_tag) | IEntity | Get a reference to an entity in the entity list based on it’s tag. Will return the first entity in the list with the tag specified. If no entity is found with the specified tag, returns null. |
| Initialize() | void | Implemented as part of the IGameObject interface. Initializes the Entity list. |
| LoadContent(ContentManager Content) | void | Implemented as part of the IGameObject interface. Calls LoadContent(ContentManager Content) on all entities in the IEntity list, first casting them as type IGameObject. |
| RemoveEntity(IEntity entity) | void | Removes specified entity from the IEntity list. |
| UnloadContent() | void | Calls Unload() on the ContentManager. This should only ever be run when the game ends. |
| Update(GameTime gameTime) | void | Implemented as part of the IGameObject interface. Calls the Update(GameTime gameTime) method on all entities in the IEntity list. |

### 4.2 Entity

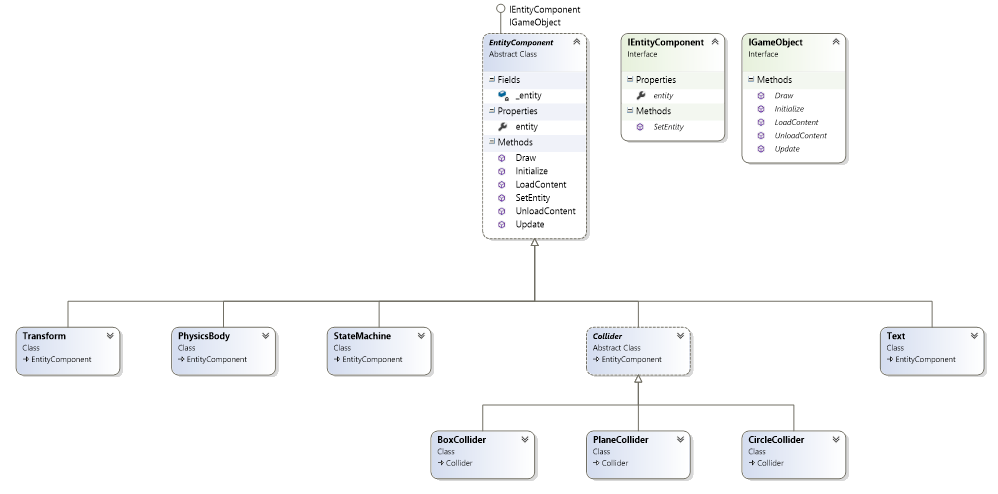
Almost everything that the user will see is an Entity. Entities store a list of EntityComponents, which implement generic behaviours that many entities may want to implement. Implements IEntity, ICollisionListener and IGameObject interfaces.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| components | List<IEntityComponent> | List of Entity Components implemented on this Entity. Used to Initialize, Load, Unload, Draw and Update components. Use GetComponent<T>() to get a reference to a specific component. |

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Name | **Data Type** | **Description** |
| tag | string | A String which can be used to identify an Entity by the EntityManager. Public Property. Implemented as part of the IEntity interface. |
| Transform | Transform | Public property, gets a reference to this entity’s Transform component. Implemented as part of the IEntity interface. |
| Methods | | |
| Name | **Return Type** | **Description** |
| AddComponent<T>() where T: IEntityComponent, new() | IEntityComponent | Adds a new IEntityComponent to the components list. Also returns a reference to the newly added IEntityComponent so assignment to a specific variable can be done when a component is added. Tells the component which Entity owns it. Implemented as part of the IEntity interface. |
| GetComponent<T>() where T: IEntityComponent | IEntityComponent | Gets a reference to EntityComponent of type T in the components list. This value must be cast to the appropriate type, as method returns IEntityComponent. Where possible, references to components should be stored in a variable and then used, rather than GetComponent<T>() being called every time a component is needed. Implemented as part of the IEntity Interface. |
| Initialize() | void | Initializes the Entity, and component list. Implemented as part of the IGameObject interface. |
| Draw(SpriteBatch spriteBatch) | void | Draws the entity, and all of its components. Image or Animation members are required for anything to actually be drawn. Implemented as part of the IGameObject interface. |
| LoadContent(ContentManager Content) | void | Used to LoadContent for this entity. Use this to load any sprites, textures, sounds, etc. Calls LoadContent(ContentManager Content) on all EntityComponents. Implemented as part of the IGameObject interface. |
| OnCollision(Collider other, Vector2 normal) + Overloads | void | Called by Collider components when a Collision has been detected. Use these methods to handle collision resolution. Implemented as part of the ICollisionListener interface. |
| UnloadContent() | void | Calls Unload() on the ContentManager. This should only ever be run when the game ends. Implemented as part of the IGameObject interface. |
| Update(GameTime gameTime) | void | Calls Update(GameTime gameTime) on all EntityComponents. Put behaviour that needs to happen every frame or so in here. Implemented as part of the IGameObject interface. |

## EntityComponent Management

Entities are composed of EntityComponents. These implement generic behaviours that many Entities may want to implement, such as collision detection and physics simulation.



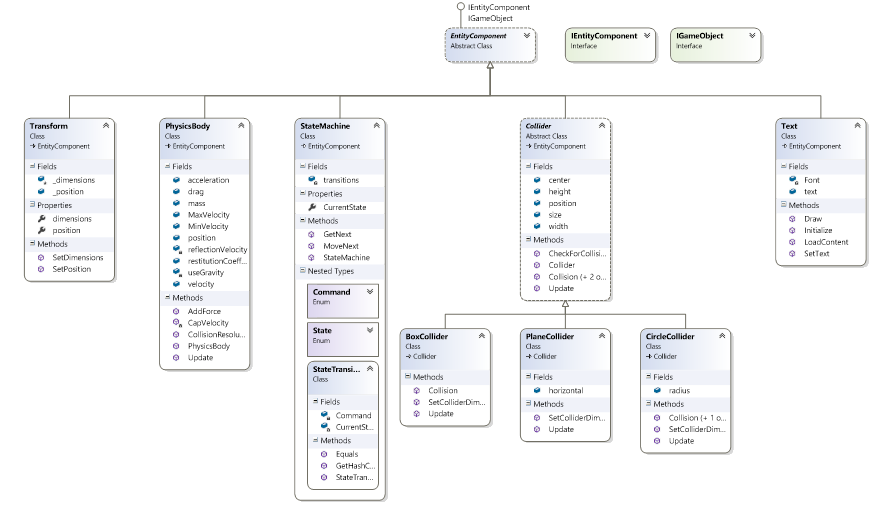
### EntityComponent

Entities are composed of components. All EntityComponents inherit from EntityComponent, which implements IEntityComponent and IGameObject.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Name | **Data Type** | **Description** |
| entity | IEntity | Public reference to the entity that this component is a component of. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Draw(SpriteBatch spriteBatch) | void | Draws the component. Image or Animation members are required for anything to actually be drawn. Implemented as part of the IGameObject interface. |
| Initialize() | void | Initializes the component. Implemented as part of the IGameObject interface. |
| LoadContent(SpriteBatch spriteBatch) | void | Used to LoadContent for this component. Use this to load any sprites, textures, sounds, etc. Implemented as part of the IGameObject interface. |
| SetEntity(IEntity entity) | void | Sets the \_entity field to specified entity. |
| UnloadContent() | void | Calls Unload() on the ContentManager. This should only ever be run when the game ends. Implemented as part of the IGameObject interface. |
| Update(GameTime gameTime) | void | Put behaviour that needs to happen every frame or so in here. Implemented as part of the IGameObject interface. |

# 5. EntityComponents



### 5.1 Transform

The Transform component handles the position and dimensions of an Entity.

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Name | **Data Type** | **Description** |
| dimensions | Vector2 | Vector2 storing the width and height of the entity. |
| position | Vector2 | Vector2 storing the x and y position of the entity. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| SetDimensions(Vector2 value) | void | Sets dimensions to specified value. |
| SetPosition(Vector2 value) | void | Sets position to specified value. |

### 5.2 Text

The Text component allows an entity to draw text to the screen.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| Font | SpriteFont | The SpriteFont used for this text. |
| text | String | The text to be displayed. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Draw(SpriteBatch spriteBatch) | void | Implemented as part of the IGameObject interface. Draws the text to the screen. |
| Initialize() | void | Implemented as part of the IGameObject interface. Initializes the EntityComponent. |
| LoadContent(ContentManager content) | void | Loads the SpriteFont. Implemented as part of the IGameObject interface. |
| SetText(string textValue) | void | Sets the text field to be equal to the parameter. |

## 5.3 State Driven Behaviours

### 5.3.1 StateMachine

The StateMachine component stores the state of the entity, and allows other objects to get a reference to the entity’s current state. This is used to perform State Driven Behaviours, such as Animation or Sound. The State Machine also includes a nested class, State Transition.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| transitions | Dictionary<StateTransition, State> | Dictionary storing possible transitions between states. |

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Name | **Data Type** | **Description** |
| CurrentState | State | This state machines current state. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| GetNext(Command command) | State | Gets a reference to the next state, based on the command that is passed in. If transition is not valid, will return an Invalid Transition exception. |
| MoveNext(Command command) | State | Sets CurrentState to GetNext(command). If transition is invalid, will return an Invalid Transition exception. |

### 5.3.2 StateTransition

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| CurrentState | State | Used as key for the dictionary. |
| Command | Command | Used as value for the dictionary. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Equals() | Bool | Override of Equals. Ensures equality of keys. |
| GetHashCode() | Int | Override of GetHashCode. Dictionary uses a combination of a key and a value, using HashCode. A combination of a key and value must always return the same hash code, so GetHashCode has been overridden. |

## 5.4 Collision Detection

Collision Detection is performed through the use of the Collider classes. The base Collider contains behaviour that all colliders will require, and is abstract. BoxCollider, CircleCollider and PlaneCollider all handle various forms of collision detection. Circles and Planes can collide, as well as Circles and other Circles. At this time, BoxColliders can only collide with other BoxColliders.

### 5.4.1 Collider

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| center | Vector2 | The position of the center of this collider. |
| height | Float | The height of this collider. In the context of circles, this is the height of the entity with the collider. |
| position | Vector2 | Position of the top left point of the collider. |
| size | Vector2 | Vector2 containing the width and height of the collider. |
| width | Float | Width of this collider. In the context of circles, this is the width of the entity with the collider. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| CheckForCollisions() | void | Collider checks against every other collider in a scene for collisions. This method gets a reference to each object in the scene, and then calls Collision, passing in the other object, to check for a collision. |
| Collision(Collider other) + Overloads | void | Depending on the collider passed in, this method will check for a collision with the other collider. This method has no implementation on the base class Collider. |

### 5.4.2 BoxCollider

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Collision(Collider other) + overloads | void | Depending on the collider passed in, this method will check for a collision with the other collider. Uses AABB collision detection by checking if the distance between the centers of the colliders is greater than the half size. |
| SetColliderDimensions(float \_width, float \_height) | void | Sets the height and width of this collider. |

### 5.4.3 CircleCollider

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| radius | Float | Radius of this circle collider. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Collision(Collider other) + overloads | void | Depending on the collider passed in, this method will check for a collision with the other collider. Uses Circle Based collision detection, checking if the distance between two objects is greater than the combined sum of the radii. When detecting collision with a Plane, Collider checks if distance between the center and a point on the plane is greater than the radius. |
| SetColliderDimensions(float \_width, float \_height, float radius) | void | Sets the height, width, and radius. |

### 5.4.4 PlaneCollider

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| horizontal | bool | Is this Plane horizontal or vertical? There is no support for diagonal planes. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Collision(Collider other) + overloads | void | Depending on the collider passed in, this method will check for a collision with the other collider. Checks for collisions by creating a new Vector equal to the Plane’s position (far top left), shifted along the appropriate axis depending on alignment, to be equal to the colliding object on that axis. Then distance is checked to determine if the objects are colliding. |
| SetColliderDimensions(float width) | void | Sets the width of the plane. |

## 5.5 Physics

The Physics class contains only a static definition for gravity. The PhysicsBody class handles Collision Resolution and Physics Simulation.

### 5.5.1 PhysicsBody

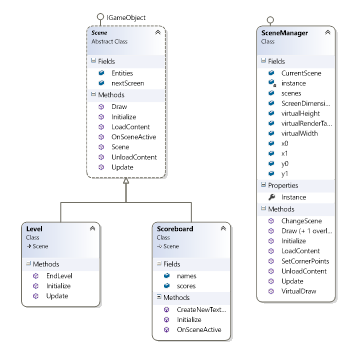
Entities with a PhysicsBody component are affected by physics. This means that they have a velocity, which can be modified by forces and acceleration, as well as drag. They have a restitution coefficient to handle their bounciness, and must obey the simulated laws of gravity.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| acceleration | Vector2 | Equal to Force/Mass. Applied to Velocity when AddForce is used. |
| drag | Float | Scalar quantity. Constantly reduces velocity. |
| mass | Float | Mass of the object. Only partially implemented, does not currently have any effect. |
| MaxVelocity | Vector2 | Maximum velocity this PhysicsBody can achieve. |
| MinVelocity | Vector2 | Maximum Negative Velocity this PhysicsBody can achieve. |
| position | Vector2 | Local reference to transform.position. |
| reflectionVelocity | Vector2 | Velocity value to be applied to velocity during collision resolution. |
| restitutionCoefficient | Float | Scalar Quantity. Used for bounciness and material simulation. |
| useGravity | Bool | Should this entity be affected by gravity? |
| velocity | Vector2 | Vector to move the attatched entity by every Update. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| AddForce(Vector2 force, ForceMode forceMode) | void | Depending on ForceMode, either applies an acceleration to the velocity of this entity, or directly changes velocity. |
| CapVelocity() | void | Clamps velocity to the min and max values. |
| CollisionResolution(CircleCollider other, Vector2 normal) + Overloads | void | Checks if the other object has a PhysicsBody. If it does, performs physics simulation between two moving objects, otherwise performs physics simulations where only one object is moving. |

# 6. Scene Management

Scene Management is done using the SceneManager and Scene classes. The SceneManager stores a list of all scenes, so that they can be initialized and have their content loaded at runtime. Only the currently active scene is updated or drawn.



### 6.1 SceneManager

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| CurrentScene | Scene | Currently active Scene. |
| scenes | List<Scene> | List of all scenes in the game build. All of these are loaded and initialized at game start. |
| ScreenDimensions | Vector2 | Height and Width of the screen. |
| virtualHeight | Float | Height of the window. |
| virtualRenderTarget | RenderTarget | Reference to the game window. |
| virtualWidth | Float | Width of the window. |

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Name | **Data Type** | **Description** |
| Instance | SceneManager | Singleton instance of SceneManager. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| ChangeScene(Scene nextScene) | void | Change the currently active scene. |
| Draw(SpriteBatch spriteBatch) | void | Calls Draw on the CurrentScene. Implemented as part of the IGameObject interface. |
| Initialize() | void | Implemented as part of the IGameObject interface. Initializes scene list. |
| LoadContent(ContentManager Content) | void | Calls LoadContent(ContentManager Content) on the CurrentScene. Implemented as part of the IGameObject interface. |
| UnloadContent() | void | Calls Unload() on the ContentManager. This should only ever be run when the game ends. Implemented as part of the IGameObject interface. |
| Update(GameTime gameTime) | void | Calls Update on the CurrentScene. Implemented as part of the IGameObject interface. |

### 6.2 Scene

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| Entities | EntityManager | This scenes EntityManager. |
| nextScreen | Scene | Reference to the next scene to be loaded. Not mandatory. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Draw(SpriteBatch spriteBatch) | void | Calls Draw on the EntityManager. Implemented as part of the IGameObject interface. |
| Initialize() | void | Implemented as part of the IGameObject interface. Initializes EntityManager. |
| LoadContent(ContentManager Content) | void | Calls LoadContent(ContentManager Content) on the EntityManager. Implemented as part of the IGameObject interface. |
| OnSceneActive() | void | Called when this scene becomes active, by the SceneManager. |
| UnloadContent() | void | Calls Unload() on the ContentManager. This should only ever be run when the game ends. Implemented as part of the IGameObject interface. |
| Update() | void | Calls Update on the EntityManager. Implemented as part of the IGameObject interface. |

# 7. Game

## 7.1 Floor and Upper Floor

The Floor and UpperFloor classes generate the floors of the level. They use the planeCollider class in order to set the colliding dimensions of a pre-set image; this allows the player / patient classes to stand on these floors without going through them. This allows the main player to traverse around the level.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| collider | PlaneCollider | The collider being used for the image |
| image | Image | The texture which is being used to provide the visual for the user, this will be drawn to the scene at the correct position required. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| UpperFloor (Vector2 position) | Vector2 | Calls method Initialize and sets the position for the Upper Floor. |
| Floor(Vector2 position) | Vector2 | Calls method Initialize and sets the position for the Floor. |
| Draw(SpriteBatch spriteBatch) | void | Calls Draw on the scene for the correct image instance with its required height and width. |
| Initialize() | void | Implemented as part of theIGameObject interface. Initializes the collider, position and specified image path to use. |
| LoadContent(ContentManagerContent) | void | CallsLoadContent(ContentManagerContent) on the image content needed and is implemented as part of the IGameObject interface. |

## 7.2 Patient

The patient class is an Entity which the player must keep alive as long as possible; it will gradually lose health over time, while it also has collision and a physical body. This entity is the main focus of the game in order for players to ‘win’. It stores a list of if statements returning its current health that causes a change in image display which is updated depending on health level, once health = 0 then the game is considered over and then moves to the Score Display.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| health | float | The health of the patient. |
| images | List<Image> | The list of all the images for the patient depending on health. |
| currentImage | Image | The current image displayed. |
| Collider | CircleCollider | The patient’s collider. |
| physicsBody | PhysicsBody | The patients physicsBody |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Initialize() | void | Initializes the Entity, health float and image list as well as the collider and physics Body. Implemented as part of the IGameObject interface. |
| Update(GameTime gameTime) | void | Lowers health every update, with elapsedTime as a modifier to make it independent of framerate. Implemented as part of the IGameObject interface. |
| OnCollision(CircleCollider other, Vector2 normal) | void | Called by Circle Collider components when a Collision has been detected. Implemented as part of the ICollisionListener interface. |
| LoadContent(ContentManager Content) | void | Loads all images. Implemented as part of the IGameObject interface. |
| Draw(SpriteBatch spriteBatch) | void | Draws the currentImage Implemented as part of the IGameObject interface |
| LowerHealth(float value) | void | Reduces health by value, then clamps it between 0 and 100. |
| SelectImage() | void | Selects which image is needed depending on the float value and will then use that image when Updated. |

## 7.3 Scoreboard

The scoreboard is used to tally up the scores earned by different players of the game depending on how much time has passed that they’ve kept all patients alive. This is done through ScoreDisplay, ScoreManager, ScoreData, and Score classes. The ScoreManager saves the scores gathered while ScoreData holds the fields required of the score (of all the time passed). Scores will only be displayed after the player has gone through the game scene and lost a patient.

## 7.4 ScoreManager

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| scoreData | List<ScoreData> | List of all the scores in the game build. If unable to create a list of ScoreData it uses back-up data. |
| fileName | String | Required to get the path of the save game played. Unable to bring forward any scores without it. Scores are saved into this file name. |

|  |  |  |
| --- | --- | --- |
| Properties | | |
| Name | **Data Type** | **Description** |
| Instance | ScoreManager | Singleton instance of StoreManager. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| ScoreManager() | void | Runs Initialize |
| Initialize() | void | Implemented as part of the IGameObject Interface. Stores all the data to the file “Scores” and saves new scores, or uses old data values. Loads the scoreData from the filename. |
| SaveHighScores(List<ScoreData> data, string fileName) | void | Gets the path of the saved file and will open it, then serializing the object into XML data allowing it to be stored and saved in the stream. |
| LoadHighScores(string fileName) | void | Calls LoadHighScores to open the recently serialized saved XML data that is read from the file. Once this is no longer needed the file is closed and the data referenced. |
| AddHighScore(string name, TimeSpan score) | void | Latest element is added to the scores list through LoadHighScores(filename) and ScoreData is then added to the current name and score data. Score is then organized accordingly and displayed by OrderByDescending. |

## 7.5 ScoreData

The values for the Score are taken from the TimeSpan, which has values for hours, minutes and seconds. TimeSpan is not serializable, so separate integers for hours, minutes and seconds are stored, which can be converted back into a TimeSpan for display. Furthermore, the TotalSeconds is stored so that the score list can easily be ordered based on who survived longest.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| PlayerName | String | Name of the player. |
| hours | int | The hours which the game has been played for. |
| minutes | int | The minutes which the game has been played for. |
| seconds | int | The seconds which the game has been played for. |
| totalSeconds | int | The total seconds which the game ended up running for. Used to order the list for the score. |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| ToTimeSpan() | void | Converts integers back into a TimeSpan for display. |

## 7.6 ScoreDisplay

ScoreDisplay is an entity whose entire purpose is to have a text component and display it.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| text | Text | Text to be used by the score and displayed |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Initialize() | void | A component class Text is added to EntityComponents |

## 7.7 Score

The Score class is the score that is displayed during gameplay, in the top centre of the screen. It is essentially a timer that counts up, letting the player know how they are doing during gameplay.

|  |  |  |
| --- | --- | --- |
| Fields | | |
| Name | **Data Type** | **Description** |
| TextDisplay | Text | Text EntityComponent. |
| value | TimeSpan | The value to be drawn. Equal to gameTime.TotalGameTime |

|  |  |  |
| --- | --- | --- |
| Methods | | |
| Name | **Return Type** | **Description** |
| Initialize() | void | Initializes the text display for the score. Implemented as part of the IGameObject interface. |
| Update(GameTime gameTime) | void | Updates the amount of game time that is passing during game play and displaying it as a string value. |