AngryTeekkaris

Generated by Doxygen 1.9.2

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 gm Namespace Reference

All types and properties of GameObjects are stored in this namespace.

Classes

- struct BlockData

  Struct for block's data.
- struct BlockMaterialData

  Struct for Block material data.
- struct BlockShapeData

  Struct for block shape data.
- struct GameObjectData

  Struct to save objects to file.
- struct PersonBody

  Struct for body of a person.
- struct PersonData

  Struct for all data needed for a person.
- struct PersonFace

  Struct for face and sound of a Teekkari or Fuksi.

Enumerations

- enum GameObjectGroup {
  background , **block** , **teekkari** , **effect** ,
  ground }

  Groups for gameobjects.

- enum GameObjectType {
  terrain1x1 , **background_tree1** , **background_tree2** , **background_lamp_pole** ,
  background_bench , **background_person1** , **background_person2** , **background_person3** ,
  block_wood1x1 , **block_metal1x1** , **block_glass1x1** , **block_plastic1x1** ,
  block_rubber1x1 , **block_concrete1x1** , **block_wood2x1** , **block_metal2x1** ,
  block_glass2x1 , **block_plastic2x1** , **block_rubber2x1** , **block_concrete2x1** ,
  block_wood2x2 , **block_metal2x2** , **block_glass2x2** , **block_plastic2x2** ,
  block_rubber2x2 , **block_concrete2x2** , **ball_wood** , **ball_metal** ,
  ball_glass , **ball_plastic** , **ball_rubber** , **ball_concrete** ,
  plank_wood , **plank_metal** , **plank_glass** , **plank_plastic** ,
  plank_rubber , **plank_concrete** , **thickplank_wood** , **thickplank_metal** ,
  thickplank_glass , **thickplank_plastic** , **thickplank_rubber** , **thickplank_concrete** ,
  prop_beer , **prop_beer_can** , **prop_chair** , **prop_table** ,
  prop_sofa2x1 , **prop_tnt** , **pickup_ik** , **pickup_sik** ,
  pickup_tefy , **pickup_tuta** , **pickup_tik** , **pickup_inkubio** ,
  pickup_kik , **pickup_professor** , **cannon** , **teekkari_ik** ,
  teekkari_sik , **teekkari_tefy** , **teekkari_tuta** , **teekkari_tik** ,
  teekkari_inkubio , **teekkari_kik** , **teekkari_professor** , **fuksi** ,
  phys_particle , **professor_particle** , **anim_effect** , **ability_cow** ,
  ability_wrench , **ability_integral** , **ground_obj** }

  A unique identifier defining the type of GameObject. All GameObjects should be spawnable without extra info.
- enum BlockMaterial {
  wood , **metal** , **glass** , **plastic** ,
  rubber , **concrete** }

  Block material properties.
- enum BlockShape {
  block_1x1 , **block_2x1** , **block_2x2** , **block_ball** ,
  block_plank , **block_thickplank** , **block_bottle** , **block_can** }

  Block shape properties.

Functions

- int **GetObjectGroup** (GameObjectType)

  Get the desired object group based on a GameObjectType.
- int **GetObjectScore** (GameObjectType type)

  Get the object score if broken, or 0 if not applicable.
- std::unique_ptr< GameObject > **IDToObject** (Game &game, GameObjectType type, float x, float y, float rot)

  Construct a child class based on GameObjectType.
- PersonData **RandomTeekkari** (GameObjectType type)

  Return a random teekkari from a guild based on GameObjectType.
- PersonData **RandomFuksi** ()

  Return a random fuksi.
- std::shared_ptr< b2Shape > **CreateShape1x1** ()
- std::shared_ptr< b2Shape > **CreateShape2x1** ()
- std::shared_ptr< b2Shape > **CreateShape2x2** ()
- std::shared_ptr< b2Shape > **CreateShapeBall** ()
- std::shared_ptr< b2Shape > **CreateShapePlank** ()
- std::shared_ptr< b2Shape > **CreateShapeThickPlank** ()
- std::shared_ptr< b2Shape > **CreateShapeBottle** ()
- std::shared_ptr< b2Shape > **CreateShapeCan** ()

Variables

- const int **objectGroupSize** = 100000000
- const std::vector< PersonFace > **teekkariHeads**

  List of teekkari heads to choose from.
- const std::vector< PersonFace > **teekkariHeads_s**

  List of teekkari heads with a different cap, because... well this is a really dumb way to accomplish this.
- const std::vector< PersonFace > **fuksiHeads**

  List of fuksi heads to choose from.
- const std::unordered_map< GameObjectType, PersonBody > **teekkariBodies**

  Lookup for teekkari bodies.
- const std::vector< gm::PersonBody > **fuksiBodies**

  List of fuksi bodies.
- const std::vector< gm::GameObjectType > **teekkaris**

  List of all teekkaris.
- const std::unordered_map< gm::GameObjectType, gm::GameObjectType > **pickupLookup**

  Lookup from pickups to teekkaris.
- const std::map< GameObjectType, BlockData > **blockTypes**

  Ordered lookup for all types of blocks.
- const std::unordered_map< BlockMaterial, BlockMaterialData > **materialProperties**

  Lookup for all block materials.
- const std::unordered_map< BlockShape, BlockShapeData > **shapeProperties**

  Lookup for all block shapes.

5.1.1 Detailed Description

All types and properties of GameObjects are stored in this namespace.

5.2 ph Namespace Reference

All game related constants are stored here.

Classes

- struct tfloat

  A struct to help with interpolating.

Functions

- float **angToRot** (float ang)

  Convert from Box2D angle (radians) to SFML rotation (degrees)
- float **rotToAng** (float rot)

  Convert from SFML rotation (degrees) to Box2D angle (radians)
- sf::Vector2f **rotateVector** (float x, float y, float rot)

  Rotate a 2D vector counterclockwise by rot degrees.

Variables

- const float **fullscreenPlayArea** = 50.0F

  Width of the area seen by a camera at zoom = 1.0F.
- const float **lightningEnergy** = 800000.0F

  Energy of a lightning, determines how many blocks it can destroy.
- const float **electricityJumpGap** = 1.0F

  The maximum gap that electricity can jump between metal blocks.
- const float **groundThickness** = 20.0F

  Thickness of the ground. More ground than this will not be visible at the lowest camera position.
- const float **cameraUpperBound** = 50.0F

  Highest coordinate visible.
- const float **groundMass** = 100.0F

  Mass of the ground object to use in calculating damage.
- const sf::Color **groundColor** = {98, 122, 31}

  Color of the ground.
- const float **gravity** = 9.81F

  The magnitude of gravity.
- const int **velocityIters** = 8

  Number of velocity iterations per b2World.Step()
- const int **positionIters** = 3

  Number of position iterations per b2World.Step()
- const float **timestep** = 0.02F

  Update is called at a rate of 1/timestep. b2World also uses this as a fixed time step.
- const float **explosionDecay** = 0.69314718F

  The decay factor lambda in exponential decay (Nt = N0 ∗ e^(-lambda ∗ t))
- const float **collisionTreshold** = 12.0F

  Impulses below this treshold don't call PhysObject OnCollision.
- const float **damageTreshold** = 30.0F

  Impulses below this don't cause damage.
- const float **soundCooldown** = 0.1F

  The minimum amount of time to wait between hitsounds.
- const float **damageScaling** = 0.18F

  A scaling factor applied to velocity when determining damage.
- const float **particleFadeTime** = 0.5F

  A particle will fade before despawning, start fading f seconds before despawn.
- const float **cannonMaxForce** = 5000

  Force of a cannon shot at maximum force.
- const float **cannonX** = -20

  The position of the cannon in the level.
- const float **personHeight** = 1.8F

  Total height of a Teekkari or Fuksi when standing upright.
- const float **personMass** = 200.0F

  Total mass of a Teekkari or Fuksi.
- const float **teekkariHP** = 8000

  HP of a Teekkari.
- const float **fuksiHP** = 200

  HP of a Fuksi.
- const int **fuksiScore** = 4000

  Score from a Fuksi.

- const int **teekkariScore** = 12000

  Score from not using a Teekkari.
- const float **pi** = 3.141592741F

  Mathematical constant pi (with max precision that float allows)
- const float **inf** = std::numeric_limits<float>::infinity()

5.2.1 Detailed Description

All game related constants are stored here.

Definition of world coordinates: The UI is defined in percentage coordinates relative to screen size using custom floats called pfloats. They were implemented by request from Ilari, based on the similar vw vh units in cascading style sheets

This means that a sprite of size 20 VW, 20 VH will be 20% of the screen width, and 20% of the screen height. This means, that despite what the size might suggest, it's shape isn't necessarily square. It is dependent on the screen aspect ratio

For most gameplay purposes however, it is important that a circle is a circle and not an oval despite what aspect ratio we are using. For this purpose, RenderSystem allows drawing sprites using either relative coordinates, or absolute coordinates (meters) These are represented by the aforementioned pfloat, and also a physics counterpart tfloat

A pfloat (percent float) has a value, and associated flag for either window width or window height

A tfloat (time float) has a previous and a current value (f0 and f1). All active modifications to it (including tfloat = tfloat assignment) always modify the current value. It also has a method Record that stores the current value to its previous (f1 => f0) This is used by RenderSystem for interpolating positions in between physics updates.

Both pfloats and tfloats can basically be used just like floats. In fact, absolutely nothing will break whether they're used or not. Object movement might simply have some stuttering.

Back to screen independent units. Since the game will also use a camera for panning, zooming etc. meters only have meaning relative to the definition of the camera. As such let us define the world, and the camera like this:

A Camera at fullscreen zoom (zoom = 1F), will see an area that is 50 meters wide around the origin.

This is defined in the constant fullscreenPlayArea. 50 meters was chosen based on Box2D's documentation of units. Based on this definition, we can define objects in meters, such as a Teekkari as 1.8 meters tall, and a box as 2 meters tall. Their distance can be 5.89 meters, and Teekkari's position can be (-9.3, 15.05) These definitions can be directly used in Box2D, and they can be directly rendered with RenderSystem.

So for example, a camera that is zoomed in by a factor of two (zoom = 0.5F) will see an area that is 25 meters wide around it.

Since camera will always draw to the full area of our target window, the shape of the camera must always be the same shape as our window or sprite shapes get distorted. This means that the height in meters that our camera sees depends on the window aspect ratio. For example, with a 1:1 aspect ratio, a fullscreen camera will see an area that is 50m x 50m. With aspect ratio 16:9, the same camera will see an area that is 50m x 28m.

The "size" of the world can be changed simply by changing the fullscreenPlayArea constant, without modifying the RenderSystem or definitions of object sizes, gravity or anything else.

Alternatively, the constant could be removed altogether, and instead the camera zoom parameter given in meters.

5.3 rng Namespace Reference

Namespace for random generator.

Functions

- void **InitializeRng** ()

  Initialize randomgenerator.
- unsigned int **RandomInt** (unsigned int min=0, int max=std::numeric_limits< unsigned int >::max())

  Get int in range [min, max] (inclusive)
- float **RandomF** ()

  Get random float between 0.0F and 1.0F.

Variables

- std::mt19937 **engine**

5.3.1 Detailed Description

Namespace for random generator.

5.4 ui Namespace Reference

Namespace for UI related constants and structs.

Classes

- struct CropArea

  a sturc for determining the area in which a shape or an elemen consisting of shapes should be rendered
- struct pfloat

  a struct for handling UI measurements in units that are relative to window height or width

Enumerations

- enum **TextAlign** { **left** , **center** , **right** }

Functions

- ui::pfloat **operator%** (const float &ff, const ui::pfloat &pp)
- ui::pfloat **operator%** (const int &ff, const ui::pfloat &pp)
- ui::pfloat **operator%** (const double &ff, const ui::pfloat &pp)
- ui::pfloat **operator∗** (const float &ff, const ui::pfloat &pp)
- ui::pfloat **operator∗** (const int &ff, const ui::pfloat &pp)
- ui::pfloat **operator∗** (const double &ff, const ui::pfloat &pp)
- ui::pfloat **operator/** (const float &ff, const ui::pfloat &pp)
- ui::pfloat **operator/** (const int &ff, const ui::pfloat &pp)
- ui::pfloat **operator/** (const double &ff, const ui::pfloat &pp)
- ui::pfloat **operator∗** (const ui::pfloat &pp, const float &ff)
- ui::pfloat **operator∗** (const ui::pfloat &pp, const int &ff)
- ui::pfloat **operator∗** (const ui::pfloat &pp, const double &ff)
- ui::pfloat **operator/** (const ui::pfloat &pp, const float &ff)
- ui::pfloat **operator/** (const ui::pfloat &pp, const int &ff)
- ui::pfloat **operator/** (const ui::pfloat &pp, const double &ff)
- float **toVHFloat** (const pfloat &p)

  convert a pfloat into a raw floating point value that is relative to window height
- float **toVWFloat** (const pfloat &p)

  convert a pfloat into a raw floating point value that is relative to window width
- CropArea **combineCropAreas** (const CropArea &a, const CropArea &b)

  combine two crop areas that fits inside both of the original crop areas

Variables

- const std::string **appName** = "AngryTeekkari"
- const std::string **appVersion** = "beta 3.7"
- const unsigned int **appMinWidth** = 400
- const unsigned int **appMinHeight** = 400
- const unsigned int **targetFramerate** = 180
- const float **targetFrametime** = 1.0F / targetFramerate
- const sf::Color **textColor** = {0, 0, 0}
- const sf::Color **buttonTextColor** = {0, 0, 0}
- const sf::Color **backgroundColor** = {255, 255, 255}
- const sf::Color **backgroundColor2** = {221, 221, 221}
- const sf::Color **buttonBackgroundColor** = {204, 204, 204}
- const sf::Color **messageBoxBackgroundColor** = {0, 0, 0, 100}
- const sf::Color **messageBoxColor** = {255, 255, 255}
- const sf::Color **highlightColor** = {200, 200, 255}
- const sf::Color **deactivatedButtonBackgroundColor** = {150, 150, 150}
- const sf::Color **inputBackgroundColor** = {204, 204, 204}
- const sf::Color **inputCaretColor** = {0, 0, 0}
- const FontID **defaultFont** = FontID::source_serif
- const FontID **defaultMonospaceFont** = FontID::consolas
- const float **defaultAbsoluteFontSize** = 16.0F
- const ui::pfloat **defaultFontSize** = (100 ∗ defaultAbsoluteFontSize / 1080) VH
- float **windowWidth**
- float **windowHeight**
- float **aspectRatio**

5.4.1 Detailed Description

Namespace for UI related constants and structs.

Chapter 6

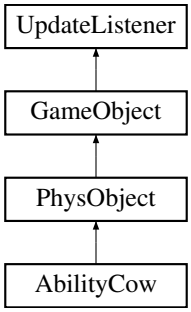
Class Documentation

6.1 AbilityCow Class Reference

Class for cow ability of an inkubioteekkari. Spawns a cow.

`#include <Teekkari.hpp>`

Inheritance diagram for AbilityCow:



Public Member Functions

- **AbilityCow** (Game &game, float x, float y, float rot)
- virtual void Render (const RenderSystem &r)

  Renders the object. Pure virtual function.
- virtual void Update ()

  Update this GameObjects position to reflect the state in the box2d world.

Protected Member Functions

- virtual void OnDeath ()

  This is called just before this object is destroyed from hp.

Protected Attributes

- float **creationTime_**

6.1.1 Detailed Description

Class for cow ability of an inkubioteekkari. Spawns a cow.

6.1.2 Member Function Documentation

6.1.2.1 OnDeath()

```
virtual void AbilityCow::OnDeath ( )  [inline], [protected], [virtual]
```

This is called just before this object is destroyed from hp.

Reimplemented from PhysObject.

6.1.2.2 Render()

```
virtual void AbilityCow::Render (
            const RenderSystem &  )  [inline], [virtual]
```

Renders the object. Pure virtual function.

Implements GameObject.

6.1.2.3 Update()

```
virtual void AbilityCow::Update ( )  [inline], [virtual]
```

Update this GameObjects position to reflect the state in the box2d world.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

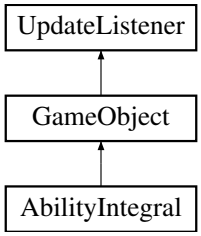
- Teekkari.hpp

6.2 AbilityIntegral Class Reference

Class for the integral ability of a professor.

`#include <Teekkari.hpp>`

Inheritance diagram for AbilityIntegral:



Public Member Functions

- **AbilityIntegral** (Game &game, float x, float y, float rot)
- virtual void Render (const RenderSystem &r)

  Renders the object. Pure virtual function.
- virtual void Update ()

Protected Member Functions

- float **GetRealTime** ()

Protected Attributes

- int **updCount_** = 0
- float **creationTime_**
- b2PolygonShape **hitShape_**
- float **height_** = 2.0F
- float **width_** = 5.15625F

6.2.1 Detailed Description

Class for the integral ability of a professor.

6.2.2 Member Function Documentation

6.2.2.1 Render()

```
virtual void AbilityIntegral::Render (
            const RenderSystem &  )  [inline], [virtual]
```

Renders the object. Pure virtual function.

Implements GameObject.

6.2.2.2 Update()

```
virtual void AbilityIntegral::Update ( )  [inline], [virtual]
```

Reimplemented from UpdateListener.

The documentation for this class was generated from the following file:

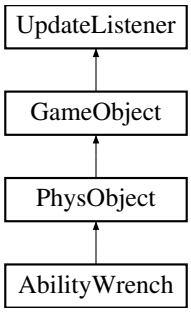
- Teekkari.hpp

6.3 AbilityWrench Class Reference

Class for wrench ability of an ikteekkari. Spawns wrenches.

```
#include <Teekkari.hpp>
```

Inheritance diagram for AbilityWrench:



Public Member Functions

- **AbilityWrench** (Game &game, float x, float y, float rot)
- virtual void Render (const RenderSystem &r)

  Renders the object. Pure virtual function.
- virtual void Update ()

  Update this GameObjects position to reflect the state in the box2d world.

Protected Member Functions

- virtual void OnDeath ()

  This is called just before this object is destroyed from hp.
- virtual void OnCollision (const b2Vec2 &velocity, PhysObject &other, const b2Contact &contact)

  OnCollision is called when this PhysObject collides with another PhysObject.

Protected Attributes

- float **creationTime_**

6.3.1 Detailed Description

Class for wrench ability of an ikteekkari. Spawns wrenches.

6.3.2 Member Function Documentation

6.3.2.1 OnCollision()

```
virtual void AbilityWrench::OnCollision (
            const b2Vec2 & relativeVelocity,
            PhysObject & other,
            const b2Contact & contact )  [inline], [protected], [virtual]
```

OnCollision is called when this PhysObject collides with another PhysObject.

Reimplemented from PhysObject.

6.3.2.2 OnDeath()

```
virtual void AbilityWrench::OnDeath ( )  [inline], [protected], [virtual]
```

This is called just before this object is destroyed from hp.

Reimplemented from PhysObject.

6.3.2.3 Render()

```
virtual void AbilityWrench::Render (
            const RenderSystem &  )  [inline], [virtual]
```

Renders the object. Pure virtual function.

Implements GameObject.

6.3.2.4 Update()

```
virtual void AbilityWrench::Update ( )  [inline], [virtual]
```

Update this GameObjects position to reflect the state in the box2d world.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

- Teekkari.hpp

6.4 Application Class Reference

The top level object of an instance.

```
#include <Application.hpp>
```

Public Member Functions

- **Application** ()
  Construct a fullscreen application and activate the first screen.
- bool **Loop** ()
  Game, event and renderloop.
- void **TransitionTo** (std::unique_ptr< Screen >)
  Transition to screen, and make it active. The old screen is destroyed.
- void **Resize** (unsigned int width, unsigned int height)
  Resize the window.
- void **Fullscreen** ()
  Go fullscreen.
- void **Exit** ()
  Close the application.
- float **GetAspectRatio** () const
  Get aspectratio.
- float **GetWindowWidth** () const
  Get window width.
- float **GetWindowHeight** () const
  Get window height.
- bool **IsFullScreen** () const
  Get if it is Fullscreen.
- AudioSystem & **GetAudioSystem** ()
  Get explicit access to Audiosystem.
- const FileManager & **GetFileManager** () const
  Get explicit access to Filemanager.
- const RenderSystem & **GetRenderSystem** () const
  Get explicit access to RenderSystem.
- const ResourceManager & **GetResourceManager** () const
  Get explicit access to RescourceManager.

6.4.1 Detailed Description

The top level object of an instance.

The documentation for this class was generated from the following file:

- Application.hpp

6.5 AudioSystem Class Reference

Framework class for playing sound effects using SoundIDs.

`#include <AudioSystem.hpp>`

Public Member Functions

- **AudioSystem** (ResourceManager &resourceManager)

  Construct an AudioSystem that queries sounds from this ResourceManager.
- void **PlaySound** (SoundID id, float volume=1.0F)

  Play the sound specified by SoundID at volume [0-1].
- void **SetGlobalVolume** (float volume)

  Set the global volume multiplier for sound effects [0-1].

6.5.1 Detailed Description

Framework class for playing sound effects using SoundIDs.

The documentation for this class was generated from the following file:

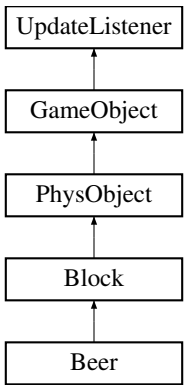
- AudioSystem.hpp

6.6 Beer Class Reference

A block that deals minus points when destroyed.

`#include <Beer.hpp>`

Inheritance diagram for Beer:



Public Member Functions

- **Beer** (Game &game, gm::GameObjectType type, float x, float y, float rot)

  Constructor.

Protected Member Functions

- virtual void OnDeath ()

  Method to be called on death. Destroys itself and causes minus points.

Additional Inherited Members

6.6.1 Detailed Description

A block that deals minus points when destroyed.

6.6.2 Member Function Documentation

6.6.2.1 OnDeath()

```
virtual void Beer::OnDeath ( )  [inline], [protected], [virtual]
```

Method to be called on death. Destroys itself and causes minus points.

Reimplemented from Block.

The documentation for this class was generated from the following file:

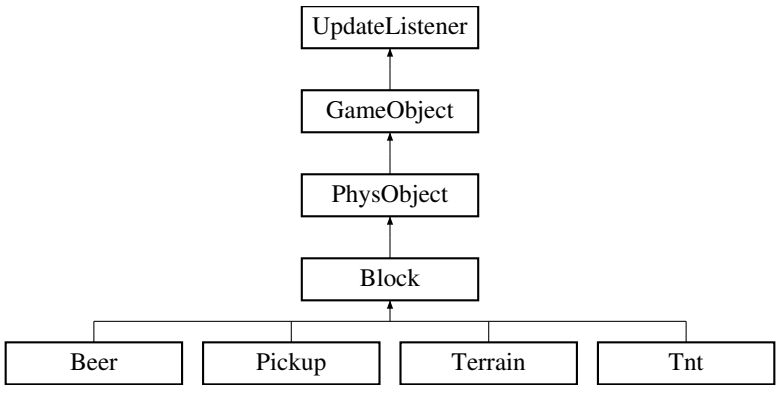
- Beer.hpp

6.7 Block Class Reference

A Block is a single body physics object with a shape and a material, that can give points when broken.

```
#include <Block.hpp>
```

Inheritance diagram for Block:



Public Member Functions

- **Block** (Game &game, gm::GameObjectType type, float x, float y, float rot)

  Construct a Block identified by this GameObjectType. It is assumed the type is a valid block.
- virtual void Render (const RenderSystem &r)

  Render this block.
- virtual void OnCollision (const b2Vec2 &velocity, PhysObject &other, const b2Contact &contact)

  Do additional things when colliding.
- virtual std::vector< sf::Sprite > GetSprites (const RenderSystem &r)

  Get all sprites of a block.
- virtual bool CheckIntersection (sf::Sprite s, const RenderSystem &r)

  Check intersection with another sprite.
- virtual std::vector< b2Body * > GetPhysBodies ()

  Get b2bodies.
- virtual bool CheckIntersection (b2Body *other)

  Check intersection with other b2body.
- const gm::BlockMaterial **GetBlockMaterial** () const

  Get block material.
- bool **ElectricityCheck** (Block &block)

  Check if electricity can flow from another block to this block.

Protected Member Functions

- virtual void OnDeath ()

  This is called just before this object is destroyed from hp.

Protected Attributes

- gm::BlockData **blockData_**
- gm::BlockShapeData **shapeData_**
- gm::BlockMaterialData **materialData_**
- float **lastHitSound_** = 0.0F

6.7.1 Detailed Description

A Block is a single body physics object with a shape and a material, that can give points when broken.

6.7.2 Member Function Documentation

6.7.2.1 CheckIntersection() [1/2]

```
virtual bool Block::CheckIntersection (
            b2Body * other )  [virtual]
```

Check intersection with other b2body.

Reimplemented from PhysObject.

6.7.2.2 CheckIntersection() [2/2]

```
virtual bool Block::CheckIntersection (
            sf::Sprite s,
            const RenderSystem & r )  [virtual]
```

Check intersection with another sprite.

Reimplemented from GameObject.

6.7.2.3 GetPhysBodies()

```
virtual std::vector< b2Body * > Block::GetPhysBodies ( )  [virtual]
```

Get b2bodies.

Reimplemented from PhysObject.

6.7.2.4 GetSprites()

```
virtual std::vector< sf::Sprite > Block::GetSprites (
            const RenderSystem & r )  [virtual]
```

Get all sprites of a block.

Reimplemented from GameObject.

6.7.2.5 OnCollision()

```
virtual void Block::OnCollision (
            const b2Vec2 & velocity,
            PhysObject & other,
            const b2Contact & contact )  [virtual]
```

Do additional things when colliding.

Reimplemented from PhysObject.

6.7.2.6 OnDeath()

```
virtual void Block::OnDeath ( )  [protected], [virtual]
```

This is called just before this object is destroyed from hp.

Reimplemented from PhysObject.

Reimplemented in Beer, Pickup, and Tnt.

6.7.2.7 Render()

```
virtual void Block::Render (
            const RenderSystem & r )  [virtual]
```

Render this block.

Implements GameObject.

The documentation for this class was generated from the following file:

- Block.hpp

6.8 gm::BlockData Struct Reference

Struct for block's data.

```
#include <GameObjectTypes.hpp>
```

Public Attributes

- std::string **blockName**
- SpriteID **sprite**
- BlockMaterial **material**
- BlockShape **shape**

6.8.1 Detailed Description

Struct for block's data.

The documentation for this struct was generated from the following file:

- GameObjectTypes.hpp

6.9 gm::BlockMaterialData Struct Reference

Struct for Block material data.

`#include <GameObjectTypes.hpp>`

Public Attributes

- BlockMaterial **material**
- float **density**
- float **friction**
- float **restitution**
- float **hpMassRatio**
- float **pointsPerMass**
- SoundID **hitSound**
- SoundID **breakSound**
- SpriteID **particle**

6.9.1 Detailed Description

Struct for Block material data.

The documentation for this struct was generated from the following file:

- GameObjectTypes.hpp

6.10 gm::BlockShapeData Struct Reference

Struct for block shape data.

`#include <GameObjectTypes.hpp>`

Public Attributes

- BlockShape **shape**
- float **volume**
- float **height**
- std::shared_ptr< b2Shape > **b2shape**
- SpriteID **halfHPSprite**
- SpriteID **lowHPSprite**

6.10.1 Detailed Description

Struct for block shape data.

The documentation for this struct was generated from the following file:

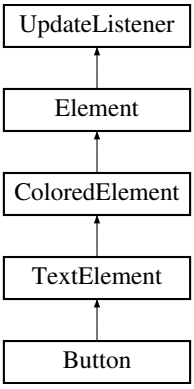
- GameObjectTypes.hpp

6.11 Button Class Reference

Class for different buttons.

`#include <Button.hpp>`

Inheritance diagram for Button:



Public Member Functions

- **Button** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- **Button** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width, const std::function< void()> mouseDownHandler)
- void **Deactivate** ()

  deactivate the button so it can't be pressed
- void **Activate** ()

  activate the button so it can be pressed again
- void **SetDeactivatedBackgroundColor** (const sf::Color &c)
- void **SetDeactivatedBackgroundColor** ()
- virtual void Render (const RenderSystem &)
- virtual void ExecuteOnMouseDown ()

  execute mouse down handler and do other things that should be done when mouse down occurs
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)

Additional Inherited Members

6.11.1 Detailed Description

Class for different buttons.

6.11.2 Member Function Documentation

6.11.2.1 ExecuteOnMouseDown()

```
virtual void Button::ExecuteOnMouseDown ( )  [virtual]
```

execute mouse down handler and do other things that should be done when mouse down occurs

screen calls this after setting focus for the element if OnMouseDown returned true

Reimplemented from Element.

6.11.2.2 OnMouseUp()

```
virtual bool Button::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Reimplemented from Element.

6.11.2.3 Render()

```
virtual void Button::Render (
            const RenderSystem &  )  [virtual]
```

Reimplemented from TextElement.

The documentation for this class was generated from the following file:

- Button.hpp

6.12 Camera Struct Reference

Struct for camera.

```
#include <Camera.hpp>
```

Public Member Functions

- void **SetFullscreen** ()

  Set camera at fullscreen zoom, and at the center of the world.

Public Attributes

- float **x** = 0

  Camera x coordinate.
- float **y** = 0

  Camera y coordinate.
- float **rot** = 0

  Camera rotation.
- float **zoom** = 1

  Camera zoom.

6.12.1 Detailed Description

Struct for camera.

A camera can be moved and zoomed, and can be used to translate object positions A Camera has a position in world space, and a zoom. zoom $<$ 1 means zooming in, zoom $>$ 1 means zooming out. zoom = 1 is fullscreen.

As per the definition in gameplay/Physics.hpp, a camera at fullscreen zoom will see an area that is 50 meters wide.

The documentation for this struct was generated from the following file:

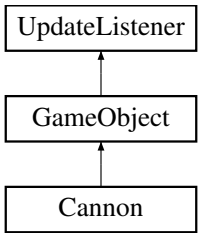
- Camera.hpp

6.13 Cannon Class Reference

Projectile (teekkari) launcher.

`#include <Cannon.hpp>`

Inheritance diagram for Cannon:



Public Member Functions

- **Cannon** (Game &game, gm::GameObjectType type, float x, float y, float rot)

  Construct a catapult at this position facing right.
- virtual ~**Cannon** ()

  Destroy underlying rigidbodies with b2dWorld.DestroyBody()
- virtual void Update ()

  Update this GameObjects position to reflect the state in the box2d world.
- virtual void Render (const RenderSystem &r)

  Render the cannon.
- virtual bool OnMouseMove (float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnMouseDown (const sf::Mouse::Button &e, float x, float y)

  Receive mouse events from the user.
- virtual bool OnMouseUp (const sf::Mouse::Button &e, float x, float y)

  Receive mouse events from the user.

Additional Inherited Members

6.13.1 Detailed Description

Projectile (teekkari) launcher.

A Cannon always aims at the mouse cursor. The launch force is based on cursor distance to the cannon

6.13.2 Member Function Documentation

6.13.2.1 OnMouseDown()

```
virtual bool Cannon::OnMouseDown (
            const sf::Mouse::Button & e,
            float x,
            float y )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

6.13.2.2 OnMouseMove()

```
virtual bool Cannon::OnMouseMove (
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

6.13.2.3 OnMouseUp()

```
virtual bool Cannon::OnMouseUp (
            const sf::Mouse::Button & e,
            float x,
            float y )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

6.13.2.4 Render()

```
virtual void Cannon::Render (
            const RenderSystem & r )  [virtual]
```

Render the cannon.

Implements GameObject.

6.13.2.5 Update()

```
virtual void Cannon::Update ( )  [virtual]
```

Update this GameObjects position to reflect the state in the box2d world.

Reimplemented from UpdateListener.

The documentation for this class was generated from the following file:

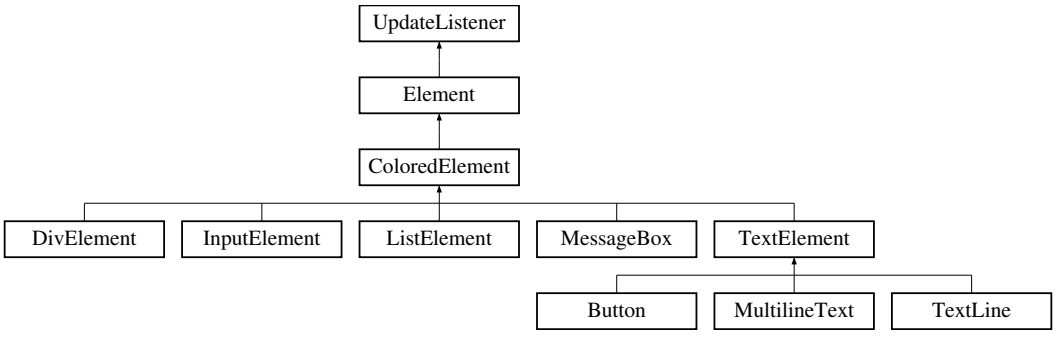
- Cannon.hpp

6.14 ColoredElement Class Reference

a simple element with a background color

```
#include <ColoredElement.hpp>
```

Inheritance diagram for ColoredElement:



Public Member Functions

- **ColoredElement** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- void **SetBackgroundColor** (const sf::Color &c)
- void **SetBackgroundColor** ()
- virtual void Render (const RenderSystem &r)

Protected Attributes

- sf::Color **backgroundColor_** = ui::backgroundColor
- sf::Color **defaultBackgroundColor_** = ui::backgroundColor

Additional Inherited Members

6.14.1 Detailed Description

a simple element with a background color

6.14.2 Member Function Documentation

6.14.2.1 Render()

```
virtual void ColoredElement::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Implements Element.

The documentation for this class was generated from the following file:

- ColoredElement.hpp

6.15 ui::CropArea Struct Reference

a sturc for determining the area in which a shape or an elemen consisting of shapes should be rendered

```
#include <UIConstants.hpp>
```

Public Member Functions

- **CropArea** (const pfloat &t, const pfloat &l, const pfloat &h, const pfloat &w)

Public Attributes

- pfloat **top** = 0 VH
- pfloat **left** = 0 VW
- pfloat **height** = 100 VH
- pfloat **width** = 100 VW

6.15.1 Detailed Description

a sturc for determining the area in which a shape or an elemen consisting of shapes should be rendered

The documentation for this struct was generated from the following file:

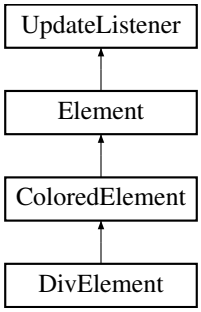
- UIConstants.hpp

6.16 DivElement Class Reference

a element for managing a large number of elements

`#include <DivElement.hpp>`

Inheritance diagram for DivElement:



Public Member Functions

- **DivElement** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- int InsertElement (std::shared_ptr< Element > element)

  insert a new element to the div
- void **RemoveElement** (int id)

  removes the element associated with the id returned by InsertElement
- std::shared_ptr< Element > **GetElement** (int id)

  returns the element associated with the id returned by InsertElement
- const std::map< int, std::shared_ptr< Element > > & **GetElements** () const
- void **ClearElements** ()

  removes all elements from the div
- virtual void SetPosition (ui::pfloat x, ui::pfloat y)
- virtual void SetTop (ui::pfloat top)
- virtual void SetLeft (ui::pfloat left)
- virtual void SetSize (ui::pfloat w, ui::pfloat h)
- virtual void SetHeight (ui::pfloat height)
- virtual void SetWidth (ui::pfloat width)
- virtual void OnWindowResize ()

  this method is called whenever the window is resized
- virtual void SetOffsetX (const ui::pfloat &ox)
- virtual void SetOffsetX ()
- virtual void SetOffsetY (const ui::pfloat &oy)

- virtual void SetOffsetY ()
- virtual void SetCropArea (const ui::CropArea &a)

  set the area in which the element must be rendered
- virtual void SetCropArea ()
- virtual void Hide ()

  hide the element and all its child elements
- virtual void Show ()

  show the element and all its child elements

Additional Inherited Members

6.16.1 Detailed Description

a element for managing a large number of elements

makes the positions of child elements relative to its own position

6.16.2 Member Function Documentation

6.16.2.1 Hide()

```
virtual void DivElement::Hide ( )  [virtual]
```

hide the element and all its child elements

Reimplemented from Element.

6.16.2.2 InsertElement()

```
int DivElement::InsertElement (
            std::shared_ptr< Element > element )
```

insert a new element to the div

Returns

an integer id that can be used later on to access or remove the element

6.16.2.3 OnWindowResize()

```
virtual void DivElement::OnWindowResize ( )  [virtual]
```

this method is called whenever the window is resized

Reimplemented from Element.

6.16.2.4 SetCropArea() [1/2]

```
virtual void DivElement::SetCropArea ( )  [virtual]
```

Reimplemented from Element.

6.16.2.5 SetCropArea() [2/2]

```
virtual void DivElement::SetCropArea (
            const ui::CropArea & a )  [virtual]
```

set the area in which the element must be rendered

Reimplemented from Element.

6.16.2.6 SetHeight()

```
virtual void DivElement::SetHeight (
            ui::pfloat height )  [virtual]
```

Reimplemented from Element.

6.16.2.7 SetLeft()

```
virtual void DivElement::SetLeft (
            ui::pfloat left )  [virtual]
```

Reimplemented from Element.

6.16.2.8 SetOffsetX() [1/2]

```
virtual void DivElement::SetOffsetX ( )  [virtual]
```

Reimplemented from Element.

6.16.2.9 SetOffsetX() [2/2]

```
virtual void DivElement::SetOffsetX (
            const ui::pfloat & ox )  [virtual]
```

Reimplemented from Element.

6.16.2.10 SetOffsetY() [1/2]

```
virtual void DivElement::SetOffsetY ( )  [virtual]
```

Reimplemented from Element.

6.16.2.11 SetOffsetY() [2/2]

```
virtual void DivElement::SetOffsetY (
            const ui::pfloat & oy )  [virtual]
```

Reimplemented from Element.

6.16.2.12 SetPosition()

```
virtual void DivElement::SetPosition (
            ui::pfloat x,
            ui::pfloat y )  [virtual]
```

Reimplemented from Element.

6.16.2.13 SetSize()

```
virtual void DivElement::SetSize (
            ui::pfloat w,
            ui::pfloat h )  [virtual]
```

Reimplemented from Element.

6.16.2.14 SetTop()

```
virtual void DivElement::SetTop (
            ui::pfloat top )  [virtual]
```

Reimplemented from Element.

6.16.2.15 SetWidth()

```
virtual void DivElement::SetWidth (
            ui::pfloat width )  [virtual]
```

Reimplemented from Element.

6.16.2.16 Show()

```
virtual void DivElement::Show ( )  [virtual]
```

show the element and all its child elements

Reimplemented from Element.

The documentation for this class was generated from the following file:

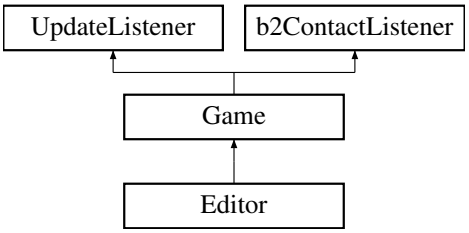
- DivElement.hpp

6.17 Editor Class Reference

Class for the game editor.

```
#include <Editor.hpp>
```

Inheritance diagram for Editor:



Public Member Functions

- **Editor** (GameScreen &s, Level level)

  Constructor.
- void **SetSelectedElement** (gm::GameObjectType t)

  Ui uses this to report the block/element the player wants to spawn next to the level.
- void **AddProjectile** (gm::GameObjectType t)

  Add the projectile to starting projectile list and call GameScreen::UpdateProjectileList()
- void **RemoveProjectile** (std::size_t index)

  Remove the element at the given idex in projectile list and call GameScreen::UpdateProjectileList()
- virtual void Resume ()

  Resume to the game and activate physics.
- virtual void **Restart** ()

  Restart the game and deactive physics.
- bool **InPlayMode** () const

  Returns if playmode is active.
- virtual bool OnMouseMove (float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)

  Receive keyboard events from the user.
- void **Play** ()

  Activate game.
- Level & **GetLevel** ()

  Get level.
- void **SaveLevel** ()

  Save changes to the level stored in the game object.
- virtual bool IsEditor () const

  Determines if it is editor. Returns always true.

Additional Inherited Members

6.17.1 Detailed Description

Class for the game editor.

6.17.2 Member Function Documentation

6.17.2.1 IsEditor()

```
virtual bool Editor::IsEditor ( ) const  [inline], [virtual]
```

Determines if it is editor. Returns always true.

Reimplemented from Game.

6.17.2.2 OnKeyDown()

```
virtual bool Editor::OnKeyDown (
            const sf::Event::KeyEvent &  )  [virtual]
```

Receive keyboard events from the user.

Reimplemented from UpdateListener.

6.17.2.3 OnMouseDown()

```
virtual bool Editor::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from Game.

6.17.2.4 OnMouseMove()

```
virtual bool Editor::OnMouseMove (
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from Game.

6.17.2.5 OnMouseUp()

```
virtual bool Editor::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from Game.

6.17.2.6 Resume()

```
virtual void Editor::Resume ( )  [virtual]
```

Resume to the game and activate physics.

Reimplemented from Game.

The documentation for this class was generated from the following file:

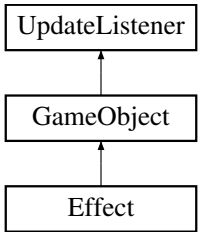
- Editor.hpp

6.18 Effect Class Reference

Effects are visible objects that don't have physical effects.

```
#include <Effect.hpp>
```

Inheritance diagram for Effect:



Public Member Functions

- **Effect** (Game &game, AnimationID anim, float x, float y, float rot, float size=1.0F, float fps=24.0F, float duration=1.0F, bool loop=false)

  Constructor.
- int **GetFrame** ()

  Get frame.
- bool CheckDuration ()

  Check if effect has ended.
- virtual void Render (const RenderSystem &r)

  Renders the effect.
- virtual void Update ()

  Updates effect.

Additional Inherited Members

6.18.1 Detailed Description

Effects are visible objects that don't have physical effects.

6.18.2 Member Function Documentation

6.18.2.1 CheckDuration()

```
bool Effect::CheckDuration ( )  [inline]
```

Check if effect has ended.

For checking if effect has ended

6.18.2.2 Render()

```
virtual void Effect::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the effect.

Implements GameObject.

6.18.2.3 Update()

```
virtual void Effect::Update ( )  [inline], [virtual]
```

Updates effect.

Reimplemented from UpdateListener.

The documentation for this class was generated from the following file:

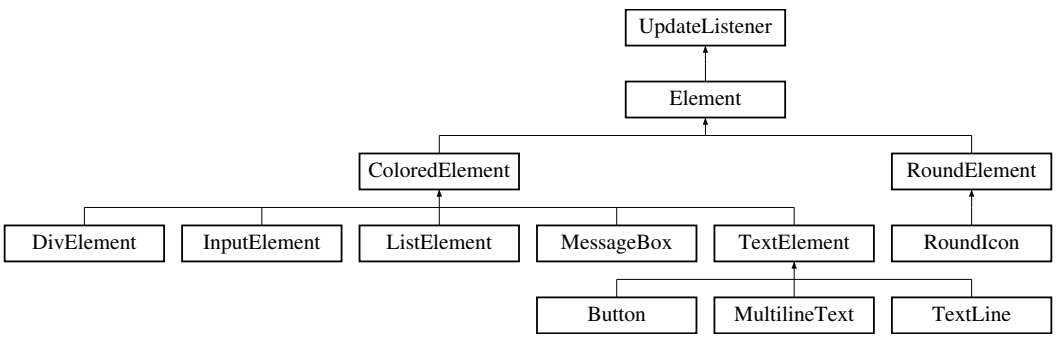
- Effect.hpp

6.19 Element Class Reference

Base class for elements.

```
#include <Element.hpp>
```

Inheritance diagram for Element:



Public Member Functions

- **Element** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- virtual void Render (const RenderSystem &)=0
- virtual void **SetPosition** (ui::pfloat x, ui::pfloat y)
- virtual void **SetTop** (ui::pfloat top)
- ui::pfloat **GetTopY** () const

  returns the raw y coordinate of the element without considering the offset
- virtual void **SetLeft** (ui::pfloat left)
- ui::pfloat **GetLeftX** () const

  returns the raw x coordinate of the element without considering the offset
- virtual void **SetSize** (ui::pfloat w, ui::pfloat h)
- virtual void **SetHeight** (ui::pfloat height)
- ui::pfloat **GetHeight** () const
- virtual void **SetWidth** (ui::pfloat width)
- ui::pfloat **GetWidth** () const
- virtual bool isInside (float xw, float yh) const

  checks if the given coordinates are inside the elements and its crop area and the element is visible
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)

  return true if the element captures the event, but doesn't execute any event handlers yet
- bool ClickSoundShouldBePlayed () const

  return true if screen should play click sound after this element is clicked
- virtual void ExecuteOnMouseDown ()

  execute event handlers and do all other things that must be done on the event occurs
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)
- virtual bool OnMouseMove (float xw, float yh)
- virtual bool OnMouseScroll (float delta, float xw, float yh)
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)
- virtual bool OnKeyUp (const sf::Event::KeyEvent &)
- virtual bool OnTextEntered (const sf::Event::TextEvent &)
- virtual void OnWindowResize ()

  this method is called whenever the window is resized
- void **SetMouseDownHandler** (const std::function< void()> f)
- void **SetMouseDownHandler** ()
- void **SetMouseUpHandler** (const std::function< void()> f)
- void **SetMouseUpHandler** ()
- void **SetMouseEnterHandler** (const std::function< void()> f)
- void **SetMOuseEnterHandler** ()
- void **SetMouseLeaveHandler** (const std::function< void()> f)
- void **SetMouseLeaveHandler** ()
- void SetMouseScrollHandler (const std::function< void(float delta)> f)
- void **SetMouseScrollHandler** ()
- void **SetFocusChangeHandler** (const std::function< void(bool focused)> f)
- void **SetFocusChangeHandler** ()
- void **SetWindowResizeHandler** (const std::function< void()> f)
- void **SetWindowResizeHandler** ()
- virtual void **Blur** ()

  remove frocus from the element
- virtual void **Focus** ()

  give focus for the element
- virtual void **SetOffsetX** (const ui::pfloat &ox)
- virtual void **SetOffsetX** ()
- virtual void **SetOffsetY** (const ui::pfloat &oy)

- virtual void **SetOffsetY** ()
- virtual void SetCropArea (const ui::CropArea &a)

  set the area in which the element must be rendered
- virtual void **SetCropArea** ()
- bool **IsCropped** () const
- ui::CropArea **GetCropArea** () const
- ui::pfloat **toVH** (const ui::pfloat &) const

  convert any pfloat into a pfloat that is relative to window height
- ui::pfloat **toVW** (const ui::pfloat &) const

  convert any pfloat into a pfloat that is relative to window width
- ui::pfloat **GetTop** () const

  get the y coordinate in which the offset has been taken into account
- ui::pfloat **GetLeft** () const

  get the x coordinate in which the offset has been taken into account
- void **SetFocusCapture** (bool b)

  tell the element to either capture or pass events that can give it a focus
- void SetTitle (const std::string &s)

  set a HTML-like title for the elemnt
- bool **IsVisible** () const
- virtual void Hide ()
- virtual void Show ()

Protected Member Functions

- bool **isInsideCropArea** (float xvw, float yvh) const
- void **RenderTitle** (const RenderSystem &r)

Protected Attributes

- ui::pfloat **x_**
- ui::pfloat **y_**
- ui::pfloat **w_**
- ui::pfloat **h_**
- ui::pfloat **offsetX_** = 0 VW
- ui::pfloat **offsetY_** = 0 VH
- bool **captureFocus_** = false
- bool **visible_** = true
- std::function< void()> **mouseDownHandler_** = NULL
- std::function< void()> **mouseUpHandler_** = NULL
- std::function< void()> **mouseEnterHandler_** = NULL
- std::function< void()> **mouseLeaveHandler_** = NULL
- std::function< void(float delta)> **mouseScrollHandler_** = NULL
- std::function< void(bool focused)> **focusChangeHandler_** = NULL
- std::function< void()> **windowResizeHandler_** = NULL
- bool **mouseIn_** = false
- bool **focused_** = false
- bool **cropped_** = false
- ui::CropArea **cropArea_**
- std::string **title_** = ""
- ui::pfloat **titleFontSize_** = ui::defaultFontSize
- ui::pfloat **titleX_**
- ui::pfloat **titleY_**
- ui::pfloat **titleW_** = 1 VW
- bool **renderTitle** = false

6.19.1 Detailed Description

Base class for elements.

6.19.2 Member Function Documentation

6.19.2.1 ClickSoundShouldBePlayed()

```
bool Element::ClickSoundShouldBePlayed ( ) const
```

return true if screen should play click sound after this element is clicked

this method is called only if OnMouseDown returned true

6.19.2.2 ExecuteOnMouseDown()

```
virtual void Element::ExecuteOnMouseDown ( )  [virtual]
```

execute event handlers and do all other things that must be done on the event occurs

this method is called only if OnMouseDown returned true

Reimplemented in Button.

6.19.2.3 Hide()

```
virtual void Element::Hide ( )  [virtual]
```

Reimplemented in DivElement.

6.19.2.4 isInside()

```
virtual bool Element::isInside (
            float xw,
            float yh ) const  [virtual]
```

checks if the given coordinates are inside the elements and its crop area and the element is visible

Reimplemented in RoundElement.

6.19.2.5 OnKeyDown()

```
virtual bool Element::OnKeyDown (
            const sf::Event::KeyEvent &  )  [inline], [virtual]
```

Reimplemented from UpdateListener.

6.19.2.6 OnKeyUp()

```
virtual bool Element::OnKeyUp (
            const sf::Event::KeyEvent &  )  [inline], [virtual]
```

Reimplemented from UpdateListener.

6.19.2.7 OnMouseDown()

```
virtual bool Element::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

return true if the element captures the event, but doesn't execute any event handlers yet

ideally this should be a pure function

Reimplemented from UpdateListener.

Reimplemented in MessageBox.

6.19.2.8 OnMouseMove()

```
virtual bool Element::OnMouseMove (
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.19.2.9 OnMouseScroll()

```
virtual bool Element::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.19.2.10 OnMouseUp()

```
virtual bool Element::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.19.2.11 OnTextEntered()

```
virtual bool Element::OnTextEntered (
            const sf::Event::TextEvent &  )  [inline], [virtual]
```

Reimplemented from UpdateListener.

6.19.2.12 OnWindowResize()

```
virtual void Element::OnWindowResize ( )  [virtual]
```

this method is called whenever the window is resized

Reimplemented in DivElement, InputElement, and ListElement.

6.19.2.13 Render()

```
virtual void Element::Render (
            const RenderSystem &  )  [pure virtual]
```

Reimplemented from UpdateListener.

6.19.2.14 SetCropArea()

```
virtual void Element::SetCropArea (
            const ui::CropArea & a )  [inline], [virtual]
```

set the area in which the element must be rendered

Reimplemented in DivElement, InputElement, and ListElement.

6.19.2.15 SetMouseScrollHandler()

```
void Element::SetMouseScrollHandler (
            const std::function< void(float delta)> f )  [inline]
```

delta is the wheel offset (positive is up/left, negative is down/right).

6.19.2.16 SetTitle()

```
void Element::SetTitle (
            const std::string & s )
```

set a HTML-like title for the elemnt

currently RoundIcon is the only element that implements this functionality

6.19.2.17 Show()

```
virtual void Element::Show ( )  [virtual]
```

Reimplemented in DivElement.

The documentation for this class was generated from the following file:

- Element.hpp

6.20 FileManager Class Reference

Framework class for loading and saving data.

```
#include <FileManager.hpp>
```

Public Member Functions

- bool **LoadTexture** (sf::Texture &texture, const std::string &path) const
  Load a texture from this path, true if successful, false if not.
- bool **LoadAudio** (sf::SoundBuffer &soundBuffer, const std::string &path) const
  Load an audio clip from this path, true if successful, false if not.
- bool **LoadFont** (sf::Font &font, const std::string &path) const
  Load a font from this path, true if successful, false if not.
- std::vector< Level > **ListLevels** () const
  Return a list of all playable levels.
- std::vector< Level > **ListEndless** () const
  Return a list of levels that can extend an endless game.
- bool **SaveLevel** (Level &level) const
  Save a level. If the level already existed, this will overwrite it.
- void **DeleteLevel** (const Level &level) const
  Delete a level.

6.20.1 Detailed Description

Framework class for loading and saving data.

The documentation for this class was generated from the following file:

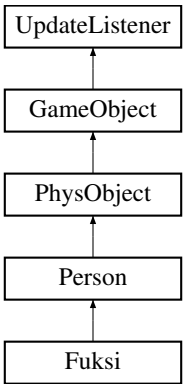
- FileManager.hpp

6.21 Fuksi Class Reference

Class for enemies.

`#include <Fuksi.hpp>`

Inheritance diagram for Fuksi:



Public Member Functions

- **Fuksi** (Game &game, float x, float y, float rot, gm::PersonData data)
  Constructor.
- **Fuksi** (Game &game, float x, float y, float rot)

Protected Member Functions

- virtual void OnDeath ()
  Increments points and other things on death.

Additional Inherited Members

6.21.1 Detailed Description

Class for enemies.

6.21.2 Member Function Documentation

6.21.2.1 OnDeath()

```
virtual void Fuksi::OnDeath ( )  [inline], [protected], [virtual]
```

Increments points and other things on death.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

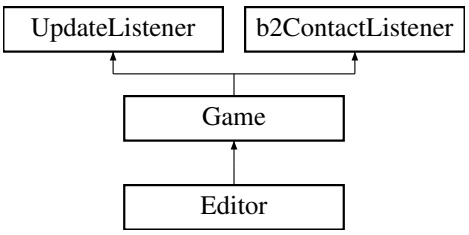
- Fuksi.hpp

6.22 Game Class Reference

Game class that manages all gameobjects and implements majority of the gamelogic.

```
#include <Game.hpp>
```

Inheritance diagram for Game:



Public Member Functions

- **Game** (GameScreen &)

  Construct a game but don't add any objects.
- **Game** (GameScreen &s, Level level)

  Construct a game, and load the provided level into it.
- virtual void Render (const RenderSystem &r)

  Render all objects in this game.
- virtual void Update ()

  Update all objects in this game.
- void **Pause** ()

  UI uses this to pause the physics simulation.
- virtual void Resume ()

  UI uses this to continue physics simulation after pausing it.
- void **Restart** ()

  Restart the game.

- void **LoadLevel** (Level level)

  Create all objects from this level.
- int **GetMaxScore** ()

  Get maxscore.
- int AddObject (std::unique_ptr< GameObject >)

  Add an existing object and take ownership. Also assign the object a gameID.
- int **CreateObject** (gm::GameObjectData data)

  Create a new GameObject from the specified data.
- int **CreateObject** (gm::GameObjectType type, float x=0, float y=0, float rot=0)

  Create a new GameObject with specified type, at this location and rotation.
- int **CreateTeekkari** (gm::PersonData data, float x=0, float y=0, float rot=0)

  Create a Teekkari from data.
- void **DestroyObject** (int id)

  Destroy the object with specified id.
- void **ClearObjects** ()

  Clear all objects.
- GameObject & **GetObject** (int id)

  Get a reference to the GameObject with this id.
- std::vector< GameObject * > **GetObjects** ()

  Get a list of all objects.
- unsigned int **GetTicks** () const

  Get the time in ticks.
- float **GetTime** () const

  Get the time in seconds.
- float **GetTimeForUI** () const

  Get the time in secods for UI purposes.
- bool **IsPaused** () const

  Is the game paused?
- bool **CannonDisabled** () const

  Returns true if cannon is disabled.
- AudioSystem & **GetAudioSystem** () const

  Get a reference to an AudioSystem to play sounds.
- b2World & **GetB2World** ()

  Get a reference to a b2World to add rigidbodies.
- GameScreen & **GetScreen** ()

  Get gamescreen.
- virtual void **BeginContact** (b2Contact *contact)

  Callback for Box2D contacts.
- const Camera & **GetCamera** () const

  Get a copy of current Camera.
- void **ResetCamera** ()

  Reset the camera to a natural fullscreen view.
- void **SetCameraPos** (float x, float y)

  Set the camera position.
- void **SetCameraZoom** (float zoom)

  Set the camera zoom.
- void **SetCameraRot** (float rot)

  Set the camera rotation.
- void **CheckLevelEnd** ()

  End the level if level is at end.
- void **AddPoints** (int p)

Increment current points.

- void **AddTeekkari** (gm::GameObjectType teekkari)

  Add a teekkari.
- void **ProfessorPause** ()

  Pause time for everyone, except the professor !
- void **ProfessorResume** ()

  Time moves again.
- void **SelectProjectile** (int index)

  UI calls this to report Game that the user has selected a projectile.
- bool **TakeProjectile** (gm::PersonData &teekkari)

  Pop out the selected teekkari. This will also update UI.
- virtual bool OnMouseMove (float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)

  Receive mouse events from the user.
- virtual bool OnMouseScroll (float delta, float xw, float yh)

  Receive mouse events from the user.
- bool **NoFuksis** ()

  Check the amount of fuksis in the level. Return true if there are none.
- bool **NoTeekkaris** ()

  Check the amount of teekkaris left. Return true if there are none.
- bool **NoActivity** ()

  Checks if there are active teekkaris, or active abilities in the level.
- virtual bool IsEditor () const

  Check if the object is an editor. Return always false.

Protected Member Functions

- void **UpdateProjectileList** ()

  Notify UI of the changes to projectiles.
- void **CheckCameraBounds** ()

  Set the camera inside world's bounds.

Protected Attributes

- GameScreen & **screen_**
- IDCounter **IDCounter_**
- std::map< int, std::unique_ptr< GameObject > > **objects_**
- Level **level_**
- Camera **camera_**
- std::vector< gm::PersonData > **teekkarisLeft_**

  List of teekkaris that can be spawned to the cannon.
- int **chosenTeekkari_** = 0
- bool **checkForFinish_** = false
- bool **professorPause_** = false

  The professors ability can stop all other objects except himself.
- bool **isPaused_** = false
- int **points_**

- unsigned int **time_** = 0
- int **levelMaxScore_** = 0
- b2World **world_**
- bool **movingCamera_** = false
- ph::tfloat **cameraGrabX_**
- ph::tfloat **cameraGrabY_**

6.22.1 Detailed Description

Game class that manages all gameobjects and implements majority of the gamelogic.

Game owns and manages all GameObjects. It also manages the box2d world, and counts ticks (Update calls) for keeping track of time.

GameObjects are kept in a map as std::unique_ptr, and ordered based on their rendering order.

When creating new objects, their id should be assigned from one of the following groups:

IDs: Backgrounds 0 - 100 000 000 Blocks 100 000 000 - 200 000 000 Teekkari 200 000 000 - 300 000 000 Effects 300 000 000 - 400 000 000 ..

the namespace gm defines a constant called objectGroupSize, and a method int GetObjectGroup(GameObject↩ Type) that returns an integer 0, 1, 2, 3 etc.

6.22.2 Member Function Documentation

6.22.2.1 AddObject()

```
int Game::AddObject (
            std::unique_ptr< GameObject >  )
```

Add an existing object and take ownership. Also assign the object a gameID.

Note

A class can construct a GameObject themselves, and add the pointer with AddObject. AddObject then needs to simply assign the object a valid gameID

CreateObject should use gm::IDToObject to create the correct subclass of GameObject based on GameObjectType. It should then add it just like with AddObject

6.22.2.2 IsEditor()

```
virtual bool Game::IsEditor ( ) const  [inline], [virtual]
```

Check if the object is an editor. Return always false.

Reimplemented in Editor.

6.22.2.3 OnMouseDown()

```
virtual bool Game::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

Reimplemented in Editor.

6.22.2.4 OnMouseMove()

```
virtual bool Game::OnMouseMove (
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

Reimplemented in Editor.

6.22.2.5 OnMouseScroll()

```
virtual bool Game::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

6.22.2.6 OnMouseUp()

```
virtual bool Game::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Receive mouse events from the user.

Reimplemented from UpdateListener.

Reimplemented in Editor.

6.22.2.7 Render()

```
virtual void Game::Render (
            const RenderSystem & r )  [virtual]
```

Render all objects in this game.

Reimplemented from UpdateListener.

6.22.2.8 Resume()

```
virtual void Game::Resume ( )  [virtual]
```

UI uses this to continue physics simulation after pausing it.

Reimplemented in Editor.

6.22.2.9 Update()

```
virtual void Game::Update ( )  [virtual]
```

Update all objects in this game.

Reimplemented from UpdateListener.

The documentation for this class was generated from the following file:

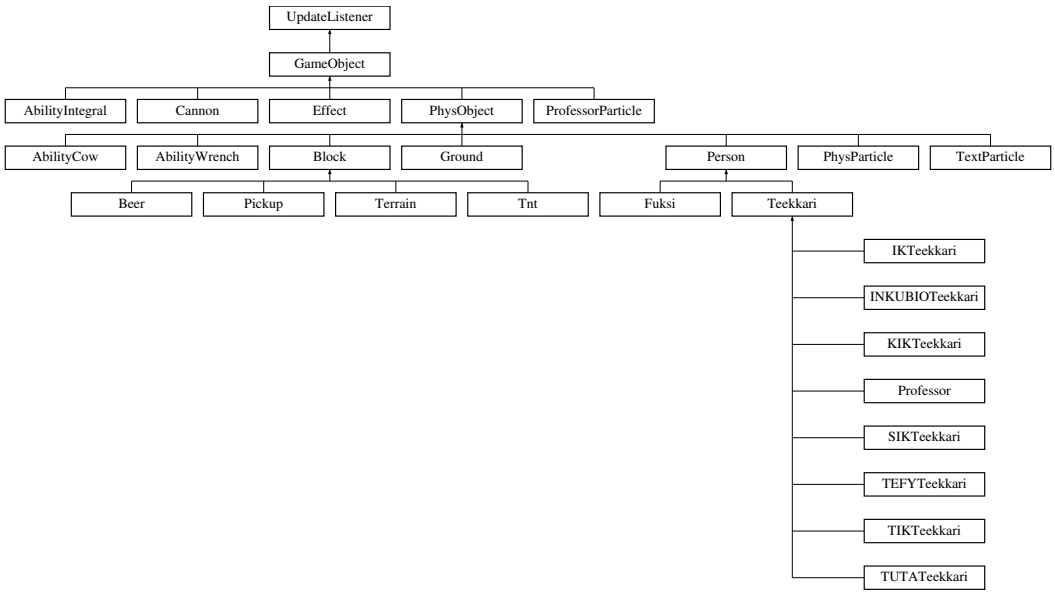
- Game.hpp

6.23 GameObject Class Reference

Base class for GameObjects.

`#include <GameObject.hpp>`

Inheritance diagram for GameObject:



Public Member Functions

- **GameObject** ()

  Initialize an empty gameobject.
- **GameObject** (Game &game, gm::GameObjectType objectType, float x, float y, float rot)

  Initialize base gameobject information.
- virtual ~**GameObject** ()=default

  Desttructor.
- virtual void Record ()

  Call record on all tfloats.
- virtual void SetX (float x)

  Set this gameobject's x.
- virtual void SetY (float y)

  Set this gameobject's y.
- virtual void SetRotation (float rot)

  Set this gameobject's rotation.
- virtual void SetPosition (float x, float y)

  Set this gameobject's position.
- virtual ph::tfloat **GetX** () const

  Get x coordinate.
- virtual ph::tfloat **GetY** () const

  Get y coordinate.
- virtual ph::tfloat **GetRot** () const

  Get rotation.
- gm::GameObjectType **GetObjectType** () const

  Get object type.
- int **GetGameID** () const

  Get gameID.
- virtual void Render (const RenderSystem &)=0

  Renders the object. Pure virtual function.
- virtual std::vector< sf::Sprite > GetSprites (const RenderSystem &r)

  Get Sprites of the object for collision test. Default an empty vector.
- virtual bool CheckIntersection (sf::Sprite s, const RenderSystem &r)

  Check intersection of this object and a sprite. Default false.
- virtual bool ContainsCoordinates (sf::Vector2f mouseCoords, const RenderSystem &r)

  Check if the object contains given relative coordinates. Default false.
- virtual std::vector< b2Body * > GetPhysBodies ()

  Get b2bodies of the object.
- virtual bool CheckIntersection (b2Body *other)

  Check intersection with another b2body.

Protected Attributes

- Game & **game_**
- ph::tfloat **x_**
- ph::tfloat **y_**
- ph::tfloat **rot_**
- gm::GameObjectType **objectType_**
- int **gameID_** = -1

Friends

- class **Game**

6.23.1 Detailed Description

Base class for GameObjects.

6.23.2 Member Function Documentation

6.23.2.1 CheckIntersection() [1/2]

```
virtual bool GameObject::CheckIntersection (
            b2Body * other )  [inline], [virtual]
```

Check intersection with another b2body.

Reimplemented in Block, Person, and PhysObject.

6.23.2.2 CheckIntersection() [2/2]

```
virtual bool GameObject::CheckIntersection (
            sf::Sprite s,
            const RenderSystem & r )  [inline], [virtual]
```

Check intersection of this object and a sprite. Default false.

Reimplemented in Block, and Person.

6.23.2.3 ContainsCoordinates()

```
virtual bool GameObject::ContainsCoordinates (
            sf::Vector2f mouseCoords,
            const RenderSystem & r )  [inline], [virtual]
```

Check if the object contains given relative coordinates. Default false.

Reimplemented in Person, and PhysObject.

6.23.2.4 GetPhysBodies()

```
virtual std::vector< b2Body * > GameObject::GetPhysBodies ( )  [inline], [virtual]
```

Get b2bodies of the object.

Reimplemented in Block, PhysParticle, TextParticle, ProfessorParticle, Person, and PhysObject.

6.23.2.5 GetSprites()

```
virtual std::vector< sf::Sprite > GameObject::GetSprites (
            const RenderSystem & r )  [inline], [virtual]
```

Get Sprites of the object for collision test. Default an empty vector.

Reimplemented in Block, and Person.

6.23.2.6 Record()

```
virtual void GameObject::Record ( )  [inline], [virtual]
```

Call record on all tfloats.

Reimplemented in Person.

6.23.2.7 Render()

```
virtual void GameObject::Render (
            const RenderSystem &  )  [pure virtual]
```

Renders the object. Pure virtual function.

Reimplemented from UpdateListener.

Implemented in Block, Cannon, Effect, Ground, PhysParticle, TextParticle, ProfessorParticle, Person, AbilityCow, AbilityWrench, AbilityIntegral, SIKTeekkari, TEFYTeekkari, and TUTATeekkari.

6.23.2.8 SetPosition()

```
virtual void GameObject::SetPosition (
            float x,
            float y )  [inline], [virtual]
```

Set this gameobject's position.

Reimplemented in Person, and PhysObject.

6.23.2.9 SetRotation()

```
virtual void GameObject::SetRotation (
            float rot )  [inline], [virtual]
```

Set this gameobject's rotation.

Reimplemented in Person, and PhysObject.

6.23.2.10 SetX()

```
virtual void GameObject::SetX (
            float x )  [inline], [virtual]
```

Set this gameobject's x.

Reimplemented in Person, and PhysObject.

6.23.2.11 SetY()

```
virtual void GameObject::SetY (
            float y )  [inline], [virtual]
```

Set this gameobject's y.

Reimplemented in Person, and PhysObject.

The documentation for this class was generated from the following file:

- GameObject.hpp

6.24 gm::GameObjectData Struct Reference

Struct to save objects to file.

```
#include <GameObjectTypes.hpp>
```

Public Attributes

- float **x**
- float **y**
- float **rot**
- GameObjectType **type**

6.24.1 Detailed Description

Struct to save objects to file.

The documentation for this struct was generated from the following file:

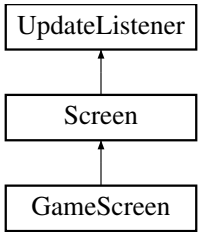
- GameObjectTypes.hpp

6.25 GameScreen Class Reference

Screen class for the game and editor.

`#include <GameScreen.hpp>`

Inheritance diagram for GameScreen:



Public Member Functions

- **GameScreen** (Application &app, const Level &initialLevel, bool editorMode=false)
  Constructor that creates a screen and starts the Game with the selected level.
- virtual void Update ()
- virtual void Render (const RenderSystem &r)
- virtual bool OnMouseDown (const sf::Mouse::Button &e, float x, float y)
- virtual bool OnMouseUp (const sf::Mouse::Button &e, float x, float y)
- virtual bool OnMouseScroll (float delta, float xw, float yh)
- virtual bool OnMouseMove (float x, float y)
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)
- virtual bool OnKeyUp (const sf::Event::KeyEvent &)
- void **Exit** ()
  exit to main menu
- void **Restart** ()
  send a restart signal for Game
- void **OnGameCompleted** (int score, int requiredMaxScore)
  show the victory message box
- void **OnGameLost** (const std::string &reason="Level failed!")
  show the game loss message box
- void **OnScoreChange** (int score)
  update the score shown in the UI
- void **UpdateProjectileList** (std::vector< std::pair< SpriteID, std::string > >)
  update the projectiles in the left hand side panel
- void **UpdateTheoreticalMaxScore** (int maxScore)

update the theoretical max score shown in the editor UI

- Game & **GetGame** ()
- Editor & **GetEditor** ()

  return a reference to the editor when the UI is in editor mode
- bool **IsInEditorMode** () const
- bool **SaveEditor** ()

  save the level that is being edited in the editor
- ui::pfloat calcTopLeftButtonLeft (unsigned char buttonNumber) const
- ui::pfloat calcTopRightLabelTop (unsigned char labelNumber) const
- ui::pfloat **calcTopRightLabelLeft** () const
- ui::pfloat **calcVictoryMessageStarTop** () const
- ui::pfloat **calcVictoryMessageStarLeft** (char starNumber) const
- ui::pfloat **calcVictoryMessageScoreTop** () const
- ui::pfloat **calcVictoryMessageContentLeft** () const
- ui::pfloat **calcVictoryMessageContentWidth** () const
- ui::pfloat **calcVictoryMessageNicknamePromptTop** () const
- ui::pfloat **calcVictoryMessageInputTop** () const
- void **saveScore** (const std::string &name, int score)

  save the player's score to the level file
- ui::pfloat **calcProjectileBarWidth** () const
- ui::pfloat **calcProjectileBarTop** () const
- ui::pfloat **calcProjectileBarBottomTop** () const
- ui::pfloat **calcProjectileBarBodyTop** () const
- ui::pfloat **calcProjectileBarBodyHeight** () const
- void **selectProjectileIcon** (std::shared_ptr< RoundIcon > i)
- void **autoSelectProjectileIcon** ()
- void **unselectProjectileIcon** ()
- ui::pfloat **calcEditorPanelLeft** () const
- ui::pfloat **calcEditorContentWidth** () const
- ui::pfloat **calcEditorContentLeft** () const
- ui::pfloat **calcEditorDropDownTop** () const
- void **addDropDownContents** (std::shared_ptr< TextElement > e)
- ui::pfloat **calcEditorMaxScoreLabelTop** () const
- ui::pfloat **calcEditorRequiredScoreLabelTop** () const
- ui::pfloat **calcEditorRequiredScoreInputTop** () const
- ui::pfloat **calcEditorTimeLimitLabelTop** () const
- ui::pfloat **calcEditorTimeLimitInputTop** () const
- ui::pfloat **calcEditorElementListTop** () const
- ui::pfloat **calcEditorElementListHeight** () const
- ui::pfloat **calcEditorPanelVisibilityButtonLeft** () const
- void **showTimeTrialOptions** ()
- void **hideTimeTrialOptions** ()
- void **setSelectedGameMode** (LevelMode m)

  sets the selected game mode in editor
- void **hideEditorPanel** ()
- void **showEditorPanel** ()

Additional Inherited Members

6.25.1 Detailed Description

Screen class for the game and editor.

6.25.2 Member Function Documentation

6.25.2.1 calcTopLeftButtonLeft()

```
ui::pfloat GameScreen::calcTopLeftButtonLeft (
            unsigned char buttonNumber ) const
```

button number is the number of the button from left starting from 1.

6.25.2.2 calcTopRightLabelTop()

```
ui::pfloat GameScreen::calcTopRightLabelTop (
            unsigned char labelNumber ) const
```

labelNumber is the number of the label from top starting from 1.

6.25.2.3 OnKeyDown()

```
virtual bool GameScreen::OnKeyDown (
            const sf::Event::KeyEvent &  )  [virtual]
```

Reimplemented from Screen.

6.25.2.4 OnKeyUp()

```
virtual bool GameScreen::OnKeyUp (
            const sf::Event::KeyEvent &  )  [virtual]
```

Reimplemented from Screen.

6.25.2.5 OnMouseDown()

```
virtual bool GameScreen::OnMouseDown (
            const sf::Mouse::Button & e,
            float x,
            float y )  [virtual]
```

Reimplemented from Screen.

6.25.2.6 OnMouseMove()

```
virtual bool GameScreen::OnMouseMove (
            float x,
            float y )  [virtual]
```

Reimplemented from Screen.

6.25.2.7 OnMouseScroll()

```
virtual bool GameScreen::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [virtual]
```

Reimplemented from Screen.

6.25.2.8 OnMouseUp()

```
virtual bool GameScreen::OnMouseUp (
            const sf::Mouse::Button & e,
            float x,
            float y )  [virtual]
```

Reimplemented from Screen.

6.25.2.9 Render()

```
virtual void GameScreen::Render (
            const RenderSystem & r )  [virtual]
```

Reimplemented from Screen.

6.25.2.10 Update()

```
virtual void GameScreen::Update ( )  [virtual]
```

Reimplemented from Screen.

The documentation for this class was generated from the following file:

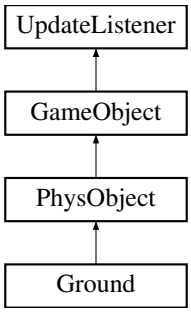
- GameScreen.hpp

6.26 Ground Class Reference

Ground class.

`#include <Ground.hpp>`

Inheritance diagram for Ground:



Public Member Functions

- **Ground** (Game &game)

  Constructs the ground according to the dimensions in physics-hpp.
- virtual float GetMass () const

  Get the mass of the ground.
- virtual void Render (const RenderSystem &r)

  Renders the ground.
- virtual void Update ()

  Update ground. Empty implementation overrides default behaviour for PhysObjects and therefore ground is not updated.

Additional Inherited Members

6.26.1 Detailed Description

Ground class.

6.26.2 Member Function Documentation

6.26.2.1 GetMass()

`virtual float Ground::GetMass ( ) const [inline], [virtual]`

Get the mass of the ground.

Reimplemented from PhysObject.

6.26.2.2 Render()

```
virtual void Ground::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the ground.

Implements GameObject.

6.26.2.3 Update()

```
virtual void Ground::Update ( )  [inline], [virtual]
```

Update ground. Empty implementation overrides default behaviour for PhysObjects and therefore ground is not updated.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

- Ground.hpp

6.27 IDCounter Struct Reference

ID-counter for gameobjects.

```
#include <Game.hpp>
```

Public Attributes

- int **backgrounds** = 0
- int **blocks** = 1 ∗ gm::objectGroupSize
- int **teekkaris** = 2 ∗ gm::objectGroupSize
- int **effects** = 3 ∗ gm::objectGroupSize

6.27.1 Detailed Description

ID-counter for gameobjects.

Each object will get a own unique id based on this counter

The documentation for this struct was generated from the following file:

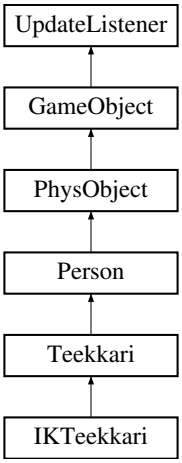
- Game.hpp

6.28 IKTeekkari Class Reference

Class for civil engineering student (IKteekkari)

`#include <Teekkari.hpp>`

Inheritance diagram for IKTeekkari:



Public Member Functions

- **IKTeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **IKTeekkari** (Game &game, float x, float y, float rot)
- virtual void OnCollision (const b2Vec2 &velocity, PhysObject &other, const b2Contact &contact)

  Handles the collision between itself and another physobject.

Protected Member Functions

- virtual void Ability (float x, float y)

Additional Inherited Members

6.28.1 Detailed Description

Class for civil engineering student (IKteekkari)

6.28.2 Member Function Documentation

6.28.2.1 Ability()

```
virtual void IKTeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

6.28.2.2 OnCollision()

```
virtual void IKTeekkari::OnCollision (
            const b2Vec2 & velocity,
            PhysObject & other,
            const b2Contact & contact )  [inline], [virtual]
```

Handles the collision between itself and another physobject.

Reimplemented from Person.

The documentation for this class was generated from the following file:

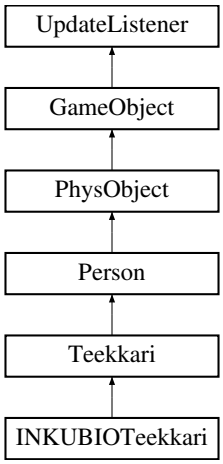
- Teekkari.hpp

6.29 INKUBIOTeekkari Class Reference

Class for bioinformation technology student (INKUBIOteekkari)

`#include <Teekkari.hpp>`

Inheritance diagram for INKUBIOTeekkari:



Public Member Functions

- **INKUBIOTeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **INKUBIOTeekkari** (Game &game, float x, float y, float rot)

Protected Member Functions

- virtual void Ability (float x, float y)

Additional Inherited Members

6.29.1 Detailed Description

Class for bioinformation technology student (INKUBIOteekkari)

6.29.2 Member Function Documentation

6.29.2.1 Ability()

```
virtual void INKUBIOTeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

The documentation for this class was generated from the following file:

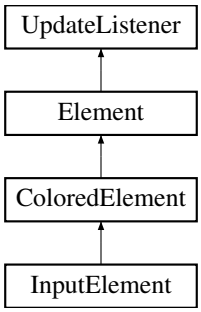
- Teekkari.hpp

6.30 InputElement Class Reference

an element that allows user to give input as a single line text

```
#include <InputElement.hpp>
```

Inheritance diagram for InputElement:



Public Member Functions

- **InputElement** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- virtual void Render (const RenderSystem &)
- void **SetText** (const std::string &)

  set the text value of the input
- std::string **GetText** () const

  get the text value of the input
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)
- virtual bool OnTextEntered (const sf::Event::TextEvent &)
- void **SetTextColor** (const sf::Color &c)
- void **SetTextColor** ()
- void **SetFont** (FontID f)
- void **SetFontSize** (const ui::pfloat &s)
- ui::pfloat **GetFontSize** ()
- virtual void SetPosition (ui::pfloat x, ui::pfloat y)
- virtual void SetTop (ui::pfloat top)
- virtual void SetLeft (ui::pfloat left)
- virtual void SetSize (ui::pfloat w, ui::pfloat h)
- virtual void SetHeight (ui::pfloat height)
- virtual void SetWidth (ui::pfloat width)
- virtual void OnWindowResize ()

  this method is called whenever the window is resized
- virtual void SetOffsetX (const ui::pfloat &ox)
- virtual void SetOffsetX ()
- virtual void SetOffsetY (const ui::pfloat &oy)
- virtual void SetOffsetY ()
- virtual void SetCropArea (const ui::CropArea &a)

  set the area in which the element must be rendered
- virtual void SetCropArea ()

Additional Inherited Members

6.30.1 Detailed Description

an element that allows user to give input as a single line text

6.30.2 Member Function Documentation

6.30.2.1 OnKeyDown()

```
virtual bool InputElement::OnKeyDown (
            const sf::Event::KeyEvent &  )  [virtual]
```

Reimplemented from Element.

6.30.2.2 OnTextEntered()

```
virtual bool InputElement::OnTextEntered (
            const sf::Event::TextEvent &  )  [virtual]
```

Reimplemented from Element.

6.30.2.3 OnWindowResize()

```
virtual void InputElement::OnWindowResize ( )  [virtual]
```

this method is called whenever the window is resized

Reimplemented from Element.

6.30.2.4 Render()

```
virtual void InputElement::Render (
            const RenderSystem &  )  [virtual]
```

Reimplemented from ColoredElement.

6.30.2.5 SetCropArea() [1/2]

```
virtual void InputElement::SetCropArea ( )  [virtual]
```

Reimplemented from Element.

6.30.2.6 SetCropArea() [2/2]

```
virtual void InputElement::SetCropArea (
            const ui::CropArea & a )  [virtual]
```

set the area in which the element must be rendered

Reimplemented from Element.

6.30.2.7 SetHeight()

```
virtual void InputElement::SetHeight (
            ui::pfloat height )  [virtual]
```

Reimplemented from Element.

6.30.2.8 SetLeft()

```
virtual void InputElement::SetLeft (
            ui::pfloat left )  [virtual]
```

Reimplemented from Element.

6.30.2.9 SetOffsetX() [1/2]

```
virtual void InputElement::SetOffsetX ( )  [virtual]
```

Reimplemented from Element.

6.30.2.10 SetOffsetX() [2/2]

```
virtual void InputElement::SetOffsetX (
            const ui::pfloat & ox )  [virtual]
```

Reimplemented from Element.

6.30.2.11 SetOffsetY() [1/2]

```
virtual void InputElement::SetOffsetY ( )  [virtual]
```

Reimplemented from Element.

6.30.2.12 SetOffsetY() [2/2]

```
virtual void InputElement::SetOffsetY (
            const ui::pfloat & oy )  [virtual]
```

Reimplemented from Element.

6.30.2.13 SetPosition()

```
virtual void InputElement::SetPosition (
            ui::pfloat x,
            ui::pfloat y )  [virtual]
```

Reimplemented from Element.

6.30.2.14 SetSize()

```
virtual void InputElement::SetSize (
            ui::pfloat w,
            ui::pfloat h )  [virtual]
```

Reimplemented from Element.

6.30.2.15 SetTop()

```
virtual void InputElement::SetTop (
            ui::pfloat top )  [virtual]
```

Reimplemented from Element.

6.30.2.16 SetWidth()

```
virtual void InputElement::SetWidth (
            ui::pfloat width )  [virtual]
```

Reimplemented from Element.

The documentation for this class was generated from the following file:

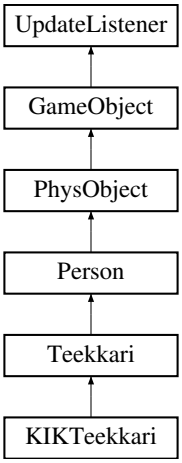
- InputElement.hpp

6.31 KIKTeekkari Class Reference

Class for mechanical engineering student (KIKteekkari)

`#include <Teekkari.hpp>`

Inheritance diagram for KIKTeekkari:



Public Member Functions

- **KIKTeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **KIKTeekkari** (Game &game, float x, float y, float rot)
- virtual void Update ()

  Updates rigidbody.

Protected Member Functions

- virtual void Ability (float x, float y)

Protected Attributes

- float **shootingInterval_** = 0.2F
- float **lastShotTime_** = 0
- int **wrenchesShot_** = 0
- b2Vec2 **targetDir** = {0, -1}

Additional Inherited Members

6.31.1 Detailed Description

Class for mechanical engineering student (KIKteekkari)

6.31.2 Member Function Documentation

6.31.2.1 Ability()

```
virtual void KIKTeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

6.31.2.2 Update()

```
virtual void KIKTeekkari::Update ( )  [inline], [virtual]
```

Updates rigidbody.

Reimplemented from Teekkari.

The documentation for this class was generated from the following file:

- Teekkari.hpp

6.32 Level Struct Reference

A struct defining the initial state of all objects at the start of a game.

```
#include <Level.hpp>
```

Public Member Functions

- int **CalculateMaxScore** ()

  Get theoretical max score of the level.

Public Attributes

- std::string **levelName** = "new level"

  Levelname.
- std::string **levelPath** = ""

  Path to the level.
- int **timeLimit** = 0

  Timelimit.
- int **perfectScore** = 0

  Score for the prefect score.
- LevelMode **levelMode** = LevelMode::normal

  Level mode of the level.
- std::vector< gm::GameObjectData > **objectData**

  Object data of the level.
- std::vector< std::pair< std::string, int > > **highscores**

  Highscores.
- SpriteID **backgroundImage** = SpriteID::background_field

  Background image.
- std::vector< gm::GameObjectType > **startingTeekkaris**

  Projectiles of the level.

6.32.1 Detailed Description

A struct defining the initial state of all objects at the start of a game.

The documentation for this struct was generated from the following file:

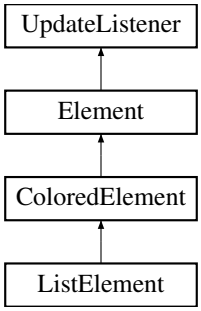
- Level.hpp

6.33 ListElement Class Reference

a list of elements that can be scrolled

`#include <ListElement.hpp>`

Inheritance diagram for ListElement:



Public Member Functions

- **ListElement** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- virtual bool OnMouseMove (float xw, float yh)
- virtual bool OnMouseScroll (float delta, float xw, float yh)
- int InsertElement (std::shared_ptr< Element > element)

  insert a new element to the list
- void **RemoveElement** (int id)

  removes the element associated with the id returned by InsertElement
- std::shared_ptr< Element > **GetElement** (int id)

  returns the element associated with the id returned by InsertElement
- void **SetSpacing** (const ui::pfloat &)

  set the amount of space shown between elements
- const std::map< int, std::shared_ptr< Element > > & GetElements () const
- void **ClearElements** ()

  removes all elements from the list
- virtual void SetPosition (ui::pfloat x, ui::pfloat y)
- virtual void SetTop (ui::pfloat top)
- virtual void SetLeft (ui::pfloat left)
- virtual void SetSize (ui::pfloat w, ui::pfloat h)
- virtual void SetHeight (ui::pfloat height)
- virtual void SetWidth (ui::pfloat width)
- virtual void OnWindowResize ()

  this method is called whenever the window is resized
- virtual void SetOffsetX (const ui::pfloat &ox)
- virtual void SetOffsetX ()
- virtual void SetOffsetY (const ui::pfloat &oy)
- virtual void SetOffsetY ()
- virtual void SetCropArea (const ui::CropArea &a)

  set the area in which the element must be rendered
- virtual void SetCropArea ()
- virtual void Hide ()
- virtual void Show ()

Additional Inherited Members

6.33.1 Detailed Description

a list of elements that can be scrolled

takes care of the positioning of the child elements

6.33.2 Member Function Documentation

6.33.2.1 GetElements()

```
const std::map< int, std::shared_ptr< Element > > & ListElement::GetElements ( ) const
```

Returns constant reference to the map of elements.

6.33.2.2 Hide()

```
virtual void ListElement::Hide ( )  [virtual]
```

Reimplemented from Element.

6.33.2.3 InsertElement()

```
int ListElement::InsertElement (
            std::shared_ptr< Element > element )
```

insert a new element to the list

Returns

an integer id that can be used later on to access or remove the element

6.33.2.4 OnMouseMove()

```
virtual bool ListElement::OnMouseMove (
            float xw,
            float yh )  [virtual]
```

Reimplemented from Element.

6.33.2.5 OnMouseScroll()

```
virtual bool ListElement::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [virtual]
```

Reimplemented from Element.

6.33.2.6 OnWindowResize()

```
virtual void ListElement::OnWindowResize ( )  [virtual]
```

this method is called whenever the window is resized

Reimplemented from Element.

6.33.2.7 SetCropArea() [1/2]

```
virtual void ListElement::SetCropArea ( )  [virtual]
```

Reimplemented from Element.

6.33.2.8 SetCropArea() [2/2]

```
virtual void ListElement::SetCropArea (
            const ui::CropArea & a )  [virtual]
```

set the area in which the element must be rendered

Reimplemented from Element.

6.33.2.9 SetHeight()

```
virtual void ListElement::SetHeight (
            ui::pfloat height )  [virtual]
```

Reimplemented from Element.

6.33.2.10 SetLeft()

```
virtual void ListElement::SetLeft (
            ui::pfloat left )  [virtual]
```

Reimplemented from Element.

6.33.2.11 SetOffsetX() [1/2]

```
virtual void ListElement::SetOffsetX ( )  [virtual]
```

Reimplemented from Element.

6.33.2.12 SetOffsetX() [2/2]

```
virtual void ListElement::SetOffsetX (
            const ui::pfloat & ox )  [virtual]
```

Reimplemented from Element.

6.33.2.13 SetOffsetY() [1/2]

```
virtual void ListElement::SetOffsetY ( )  [virtual]
```

Reimplemented from Element.

6.33.2.14 SetOffsetY() [2/2]

```
virtual void ListElement::SetOffsetY (
            const ui::pfloat & oy )  [virtual]
```

Reimplemented from Element.

6.33.2.15 SetPosition()

```
virtual void ListElement::SetPosition (
            ui::pfloat x,
            ui::pfloat y )  [virtual]
```

Reimplemented from Element.

6.33.2.16 SetSize()

```
virtual void ListElement::SetSize (
            ui::pfloat w,
            ui::pfloat h )  [virtual]
```

Reimplemented from Element.

6.33.2.17 SetTop()

```
virtual void ListElement::SetTop (
            ui::pfloat top )  [virtual]
```

Reimplemented from Element.

6.33.2.18 SetWidth()

```
virtual void ListElement::SetWidth (
            ui::pfloat width )  [virtual]
```

Reimplemented from Element.

6.33.2.19 Show()

```
virtual void ListElement::Show ( )  [virtual]
```

Reimplemented from Element.

The documentation for this class was generated from the following file:

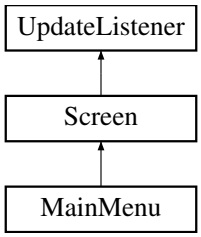
- ListElement.hpp

6.34 MainMenu Class Reference

Game's main menu.

```
#include <MainMenu.hpp>
```

Inheritance diagram for MainMenu:



Public Member Functions

- **MainMenu** (Application &app)
- virtual void Render (const RenderSystem &)
- void **SelectLevel** (const Level &level, std::weak_ptr< Button > button, int id)

  this is ment to be used only by UI call backs
- Level **GetSelectedLevel** () const

  this is ment to be used only by UI call backs
- ui::pfloat **calcListWidth** () const
- ui::pfloat **calcListElementWidth** () const
- ui::pfloat **calcRightSideElementWidth** () const
- ui::pfloat **calcListTop** () const
- ui::pfloat **calcListHeight** () const
- ui::pfloat **calcListBottomTop** () const
- ui::pfloat **calcRightSideButtonTop** (unsigned char buttonNumber) const
- ui::pfloat **calcRightSideLeft** () const
- ui::pfloat **calcScoreboardMultilineTop** () const
- ui::pfloat **calcScoreboardMultilineHeight** () const
- void **deleteSelectedLevel** ()

Additional Inherited Members

6.34.1 Detailed Description

Game's main menu.

6.34.2 Member Function Documentation

6.34.2.1 Render()

```
virtual void MainMenu::Render (
            const RenderSystem &  )  [virtual]
```

Reimplemented from Screen.

The documentation for this class was generated from the following file:

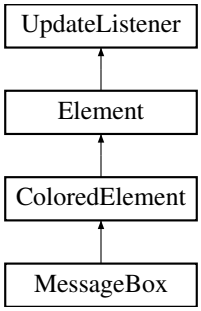
- MainMenu.hpp

6.35 MessageBox Class Reference

a simple base element for a message box

`#include <MessageBox.hpp>`

Inheritance diagram for MessageBox:



Public Member Functions

- **MessageBox** (const ui::pfloat &height, const ui::pfloat &width)
- virtual void Render (const RenderSystem &)
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)
- virtual bool OnKeyUp (const sf::Event::KeyEvent &)
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)

  return true if the element captures the event, but doesn't execute any event handlers yet
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)
- virtual bool OnMouseScroll (float delta, float xw, float yh)
- virtual bool OnTextEntered (const sf::Event::TextEvent &)

Additional Inherited Members

6.35.1 Detailed Description

a simple base element for a message box

6.35.2 Member Function Documentation

6.35.2.1 OnKeyDown()

```
virtual bool MessageBox::OnKeyDown (
            const sf::Event::KeyEvent &  )  [inline], [virtual]
```

Reimplemented from Element.

6.35.2.2 OnKeyUp()

```
virtual bool MessageBox::OnKeyUp (
            const sf::Event::KeyEvent &  )  [inline], [virtual]
```

Reimplemented from Element.

6.35.2.3 OnMouseDown()

```
virtual bool MessageBox::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

return true if the element captures the event, but doesn't execute any event handlers yet

ideally this should be a pure function

Reimplemented from Element.

6.35.2.4 OnMouseScroll()

```
virtual bool MessageBox::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [virtual]
```

Reimplemented from Element.

6.35.2.5 OnMouseUp()

```
virtual bool MessageBox::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Reimplemented from Element.

6.35.2.6 OnTextEntered()

```
virtual bool MessageBox::OnTextEntered (
            const sf::Event::TextEvent &  )  [inline], [virtual]
```

Reimplemented from Element.

6.35.2.7 Render()

```
virtual void MessageBox::Render (
            const RenderSystem &  )  [virtual]
```

Reimplemented from ColoredElement.

The documentation for this class was generated from the following file:

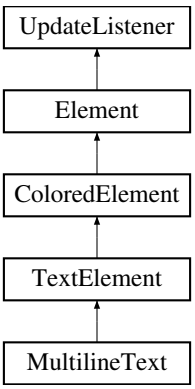
- MessageBox.hpp

6.36 MultilineText Class Reference

Element that can contain multiple lains of text.

`#include <MultilineText.hpp>`

Inheritance diagram for MultilineText:



Public Member Functions

- **MultilineText** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- virtual void SetText (const std::string &s)
- virtual void Render (const RenderSystem &r)
- void **SetRelativeLineSpacing** (const ui::pfloat &s)

  set the amount of space shown between the lines in relative units
- void **SetAbsoluteLineSpacing** (float s)

  set the amount of space shown between the lines in absolute units
- ui::pfloat **GetLineSpacing** ()

  get the amount of space shown between the lines in units relative to the current window size

Additional Inherited Members

6.36.1 Detailed Description

Element that can contain multiple lains of text.

6.36.2 Member Function Documentation

6.36.2.1 Render()

```
virtual void MultilineText::Render (
            const RenderSystem & r )  [virtual]
```

Reimplemented from TextElement.

6.36.2.2 SetText()

```
virtual void MultilineText::SetText (
            const std::string & s )  [virtual]
```

Reimplemented from TextElement.

The documentation for this class was generated from the following file:

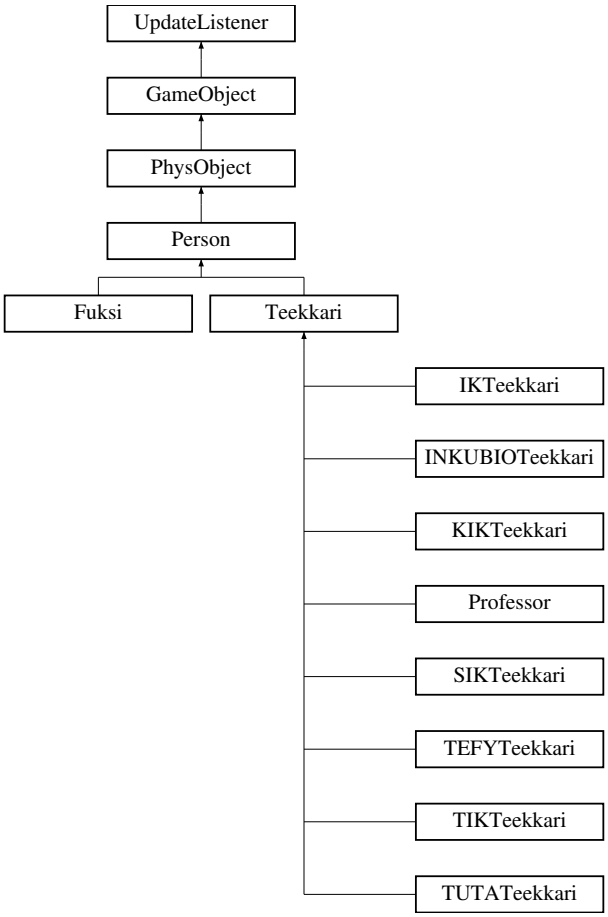
- MultilineText.hpp

6.37 Person Class Reference

Physics ragdoll that can be used for humanlike objects.

```
#include <Person.hpp>
```

Inheritance diagram for Person:



Public Member Functions

- **Person** (Game &game, gm::GameObjectType type, float x, float y, float rot, bool mirrored=false, int collision↩ Group=-5)

  Constructor. The person will point right. If mirrored = true, it will point to the left.
- virtual ~**Person** ()

  Destructor for person.
- virtual void Render (const RenderSystem &r)

  Renders the person.
- virtual float GetMass () const

  Returns mass of the person.
- virtual void Record ()

  Records all tfloats.
- virtual void Update ()

  Updates rigidbody.
- virtual void SetX (float x)

  Set x coordinate.
- virtual void SetY (float y)

  Set y coordinate.
- virtual void SetPosition (float x, float y)

  Set pos.
- virtual void SetRotation (float rot)

  Set rotation.

- virtual void OnCollision (const b2Vec2 &velocity, PhysObject &other, const b2Contact &contact)

  Handles the collision between itself and another physobject.
- virtual void Impulse (const b2Vec2 &f)

  Creates an impulse.
- virtual bool ContainsCoordinates (sf::Vector2f mouseCoords, const RenderSystem &r)

  Check if the object contains given relative coordinates. Default false.
- virtual std::vector< sf::Sprite > GetSprites (const RenderSystem &r)

  Get Sprites of the object for collision test. Default an empty vector.
- virtual bool CheckIntersection (sf::Sprite s, const RenderSystem &r)

  Check intersection of this object and a sprite. Default false.
- virtual std::vector< b2Body * > GetPhysBodies ()

  Get b2bodies of the object.
- virtual bool CheckIntersection (b2Body *other)

  Check intersection with another b2body.

Protected Attributes

- gm::PersonData **data_**
- float **lastHitSound_** = 0.0F
- b2Body * **headBody_**
- b2Body * **armRBody_**
- b2Body * **armLBody_**
- b2Body * **legRBody_**
- b2Body * **legLBody_**
- ph::tfloat **headX_**
- ph::tfloat **headY_**
- ph::tfloat **headRot_**
- ph::tfloat **armRX_**
- ph::tfloat **armRY_**
- ph::tfloat **armRRot_**
- ph::tfloat **armLX_**
- ph::tfloat **armLY_**
- ph::tfloat **armLRot_**
- ph::tfloat **legRX_**
- ph::tfloat **legRY_**
- ph::tfloat **legRRot_**
- ph::tfloat **legLX_**
- ph::tfloat **legLY_**
- ph::tfloat **legLRot_**

Static Protected Attributes

- static const float **restitution** = 0.3F
- static const float **totalHeight** = ph::personHeight
- static const float **legHeight** = 0.23913F * Person::totalHeight
- static const float **armHeight** = 1.13207F * Person::legHeight
- static const float **torsoHeight** = 1.47169F * Person::legHeight
- static const float **headHeight** = 2.064150F * Person::legHeight
- static const float **torsoWidth** = 0.8333F * Person::torsoHeight
- static const float **legWidth** = 0.6415F * Person::legHeight
- static const float **armWidth** = 0.56666F * Person::armHeight

- static const float **headWidth** = 0.89166F ∗ Person::headHeight
- static const float **torsoVolume** = Person::torsoWidth ∗ Person::torsoHeight
- static const float **legVolume** = Person::legWidth ∗ Person::legHeight
- static const float **armVolume** = Person::armWidth ∗ Person::armHeight
- static const float **headVolume** = 0.25F ∗ Person::headHeight ∗ Person::headHeight ∗ ph::pi
- static const float **totalVolume** = Person::torsoVolume + 2 ∗ Person::legVolume + 2 ∗ Person::armVolume + Person::headVolume

Additional Inherited Members

6.37.1 Detailed Description

Physics ragdoll that can be used for humanlike objects.

6.37.2 Member Function Documentation

6.37.2.1 CheckIntersection() [1/2]

```
virtual bool Person::CheckIntersection (
            b2Body * other )  [virtual]
```

Check intersection with another b2body.

Reimplemented from PhysObject.

6.37.2.2 CheckIntersection() [2/2]

```
virtual bool Person::CheckIntersection (
            sf::Sprite s,
            const RenderSystem & r )  [virtual]
```

Check intersection of this object and a sprite. Default false.

Reimplemented from GameObject.

6.37.2.3 ContainsCoordinates()

```
virtual bool Person::ContainsCoordinates (
            sf::Vector2f mouseCoords,
            const RenderSystem & r )  [virtual]
```

Check if the object contains given relative coordinates. Default false.

Reimplemented from PhysObject.

6.37.2.4 GetMass()

```
virtual float Person::GetMass ( ) const  [virtual]
```

Returns mass of the person.

Reimplemented from PhysObject.

6.37.2.5 GetPhysBodies()

```
virtual std::vector< b2Body * > Person::GetPhysBodies ( )  [virtual]
```

Get b2bodies of the object.

Reimplemented from PhysObject.

6.37.2.6 GetSprites()

```
virtual std::vector< sf::Sprite > Person::GetSprites (
            const RenderSystem & r )  [virtual]
```

Get Sprites of the object for collision test. Default an empty vector.

Reimplemented from GameObject.

6.37.2.7 Impulse()

```
virtual void Person::Impulse (
            const b2Vec2 & f )  [virtual]
```

Creates an impulse.

Reimplemented from PhysObject.

6.37.2.8 OnCollision()

```
virtual void Person::OnCollision (
            const b2Vec2 & velocity,
            PhysObject & other,
            const b2Contact & contact )  [virtual]
```

Handles the collision between itself and another physobject.

Reimplemented from PhysObject.

Reimplemented in IKTeekkari.

6.37.2.9 Record()

```
virtual void Person::Record ( )  [virtual]
```

Records all tfloats.

Reimplemented from GameObject.

6.37.2.10 Render()

```
virtual void Person::Render (
            const RenderSystem & r )  [virtual]
```

Renders the person.

Implements GameObject.

Reimplemented in SIKTeekkari, TEFYTeekkari, and TUTATeekkari.

6.37.2.11 SetPosition()

```
virtual void Person::SetPosition (
            float x,
            float y )  [virtual]
```

Set pos.

Reimplemented from PhysObject.

6.37.2.12 SetRotation()

```
virtual void Person::SetRotation (
            float rot )  [virtual]
```

Set rotation.

Reimplemented from PhysObject.

6.37.2.13 SetX()

```
virtual void Person::SetX (
            float x )  [virtual]
```

Set x coordinate.

Reimplemented from PhysObject.

6.37.2.14 SetY()

```
virtual void Person::SetY (
            float y )  [virtual]
```

Set y coordinate.

Reimplemented from PhysObject.

6.37.2.15 Update()

```
virtual void Person::Update ( )  [virtual]
```

Updates rigidbody.

Reimplemented from PhysObject.

Reimplemented in Teekkari, SIKTeekkari, TEFYTeekkari, TUTATeekkari, KIKTeekkari, and Professor.

The documentation for this class was generated from the following file:

- Person.hpp

6.38 gm::PersonBody Struct Reference

Struct for body of a person.

```
#include <GameObjectTypes.hpp>
```

Public Attributes

- SpriteID **torso** = SpriteID::torso_blue
- SpriteID **arm** = SpriteID::arm_blue
- SpriteID **leg** = SpriteID::leg_blue
- SpriteID **armb** = SpriteID::armb_blue
- std::string **guildName** = "Teemu Teekkari"

6.38.1 Detailed Description

Struct for body of a person.

The documentation for this struct was generated from the following file:

- GameObjectTypes.hpp

6.39 gm::PersonData Struct Reference

Struct for all data needed for a person.

`#include <GameObjectTypes.hpp>`

Public Attributes

- PersonFace **face**
- PersonBody **body**
- GameObjectType **objType** = GameObjectType::teekkari_ik

6.39.1 Detailed Description

Struct for all data needed for a person.

All properties needed to spawn a unique Teekkari or Fuksi. A Teekkari can be spawned with GameObjectType as well, it will have a randomly generated PersonData

The documentation for this struct was generated from the following file:

- GameObjectTypes.hpp

6.40 gm::PersonFace Struct Reference

Struct for face and sound of a Teekkari or Fuksi.

`#include <GameObjectTypes.hpp>`

Public Attributes

- SpriteID **face** = SpriteID::teekkari_head1
- SoundID **grunt** = SoundID::grunt1
- SoundID **die** = SoundID::teekkari_death1
- bool **bType** = false

6.40.1 Detailed Description

Struct for face and sound of a Teekkari or Fuksi.

The documentation for this struct was generated from the following file:

- GameObjectTypes.hpp

6.41 ui::pfloat Struct Reference

a struct for handling UI measurements in units that are relative to window height or width

`#include <UIConstants.hpp>`

Public Types

- enum **P** { **vh** , **vw** }

Public Member Functions

- **pfloat** (const float &ff, P pp)
- **operator float** () const
- pfloat **operator-** () const
- pfloat & **operator=** (const pfloat &pf)
- pfloat & **operator∗=** (const float &ff)
- pfloat & **operator/=** (const float &ff)
- pfloat & **operator+=** (const float &ff)
- pfloat & **operator-=** (const float &ff)

Public Attributes

- float **f**
- P **p**

6.41.1 Detailed Description

a struct for handling UI measurements in units that are relative to window height or width

The documentation for this struct was generated from the following file:

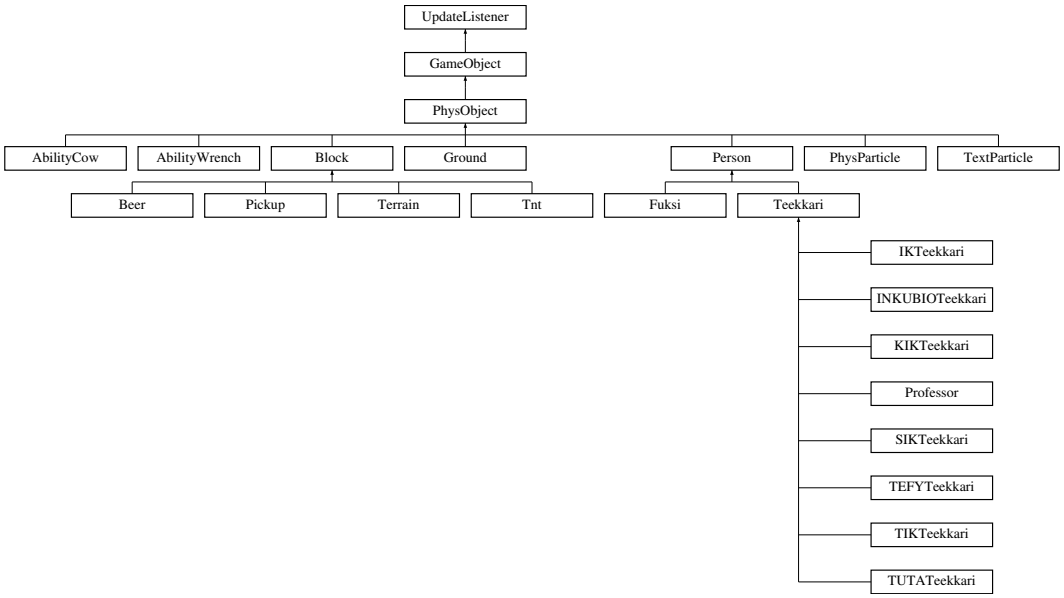
- UIConstants.hpp

6.42 PhysObject Class Reference

Class for objects that have one or more rigidbodies. Can be affected with forces and other PhysObjects.

`#include <PhysObject.hpp>`

Inheritance diagram for PhysObject:



Public Member Functions

- **PhysObject** (Game &game, gm::GameObjectType objectID, float x, float y, float rot)

  Constructor.
- virtual ~**PhysObject** ()

  Destroy underlying rigidbodies with b2dWorld.DestroyBody()
- virtual void Update ()

  Update this GameObjects position to reflect the state in the box2d world.
- virtual void OnCollision (const b2Vec2 &relativeVelocity, PhysObject &other, const b2Contact &contact)

  OnCollision is called when this PhysObject collides with another PhysObject.
- virtual void SetX (float x)

  Set this rigidbody's x.
- virtual void SetY (float y)

  Set this rigidbody's y.
- virtual void SetRotation (float rot)

  Set this rigidbody's rotation.
- virtual void SetPosition (float x, float y)

  Set this rigidbody's position.
- virtual void Impulse (const b2Vec2 &f)

  Instant change in velocity.
- virtual void **Impulse** (const b2Vec2 &f, const b2Vec2 &p)

  Instant change in velocity at point p.
- virtual void **Force** (const b2Vec2 &f)

  Force over time.

- virtual void **Force** (const b2Vec2 &f, const b2Vec2 &p)

  Force over time at point p.
- virtual void **Torque** (float t)

  Torque over time.
- virtual void **Angular** (float a)

  Instant change in angular velocity.
- virtual void **Explosion** (const b2Vec2 &center, float magnitude)

  Add explosive force away from center.
- void **ExplosionDamage** (const b2Vec2 &center, float damage)

  Deal explosive damage, that decays with distance.
- virtual void DealDamage (float damage)

  Explicitly deal damage to this objects hp.
- virtual float **GetHP** () const

  Get HP.
- virtual float GetMass () const

  Get mass.
- virtual bool ContainsCoordinates (sf::Vector2f mouseCoords, const RenderSystem &r)

  Check if the object contains given relative coordinates. Default false.
- virtual std::vector< b2Body * > GetPhysBodies ()

  Get b2bodies of the object.
- virtual bool CheckIntersection (b2Body *other)

  Check intersection with another b2body.

Protected Member Functions

- virtual void OnDeath ()

  This is called just before this object is destroyed from hp.

Protected Attributes

- b2Body * **mainBody_**

  This is the main (root) box2d body. Objects can have subparts such as bodies connected by joints.
- float **hp_** = 0
- SpriteID **hitSp_** = SpriteID::hit_stars
- b2Vec2 **hitPoint_** = {0, 0}
- bool **spawnHit_** = false

6.42.1 Detailed Description

Class for objects that have one or more rigidbodies. Can be affected with forces and other PhysObjects.

6.42.2 Member Function Documentation

6.42.2.1 CheckIntersection()

```
virtual bool PhysObject::CheckIntersection (
            b2Body * other )  [virtual]
```

Check intersection with another b2body.

Reimplemented from GameObject.

Reimplemented in Block, and Person.

6.42.2.2 ContainsCoordinates()

```
virtual bool PhysObject::ContainsCoordinates (
            sf::Vector2f mouseCoords,
            const RenderSystem & r )  [virtual]
```

Check if the object contains given relative coordinates. Default false.

Reimplemented from GameObject.

Reimplemented in Person.

6.42.2.3 DealDamage()

```
virtual void PhysObject::DealDamage (
            float damage )  [virtual]
```

Explicitly deal damage to this objects hp.

Reimplemented in Terrain.

6.42.2.4 GetMass()

```
virtual float PhysObject::GetMass ( ) const  [virtual]
```

Get mass.

Reimplemented in Ground, Person, and Terrain.

6.42.2.5 GetPhysBodies()

```
virtual std::vector< b2Body * > PhysObject::GetPhysBodies ( )  [virtual]
```

Get b2bodies of the object.

Reimplemented from GameObject.

Reimplemented in Block, PhysParticle, TextParticle, and Person.

6.42.2.6 Impulse()

```
virtual void PhysObject::Impulse (
            const b2Vec2 & f )  [virtual]
```

Instant change in velocity.

Reimplemented in Person.

6.42.2.7 OnCollision()

```
virtual void PhysObject::OnCollision (
            const b2Vec2 & relativeVelocity,
            PhysObject & other,
            const b2Contact & contact )  [virtual]
```

OnCollision is called when this PhysObject collides with another PhysObject.

Reimplemented in Block, Person, AbilityWrench, and IKTeekkari.

6.42.2.8 OnDeath()

```
virtual void PhysObject::OnDeath ( )  [inline], [protected], [virtual]
```

This is called just before this object is destroyed from hp.

Reimplemented in Beer, Block, Fuksi, Pickup, Teekkari, AbilityCow, AbilityWrench, and Tnt.

6.42.2.9 SetPosition()

```
virtual void PhysObject::SetPosition (
            float x,
            float y )  [virtual]
```

Set this rigidbody's position.

Reimplemented from GameObject.

Reimplemented in Person.

6.42.2.10 SetRotation()

```
virtual void PhysObject::SetRotation (
            float rot )  [virtual]
```

Set this rigidbody's rotation.

Reimplemented from GameObject.

Reimplemented in Person.

6.42.2.11 SetX()

```
virtual void PhysObject::SetX (
            float x )  [virtual]
```

Set this rigidbody's x.

Reimplemented from GameObject.

Reimplemented in Person.

6.42.2.12 SetY()

```
virtual void PhysObject::SetY (
            float y )  [virtual]
```

Set this rigidbody's y.

Reimplemented from GameObject.

Reimplemented in Person.

6.42.2.13 Update()

```
virtual void PhysObject::Update ( )  [virtual]
```

Update this GameObjects position to reflect the state in the box2d world.

Reimplemented from UpdateListener.

Reimplemented in Ground, PhysParticle, TextParticle, Person, Teekkari, AbilityCow, AbilityWrench, SIKTeekkari, TEFYTeekkari, TUTATeekkari, KIKTeekkari, and Professor.

The documentation for this class was generated from the following file:

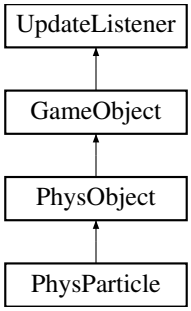
- PhysObject.hpp

6.43 PhysParticle Class Reference

Particle class with physics.

```
#include <ParticleEffect.hpp>
```

Inheritance diagram for PhysParticle:



Public Member Functions

- **PhysParticle** (Game &game, float x, float y, float rot)
  Constructor.
- virtual void Render (const RenderSystem &r)
  Renders the particle.
- virtual void Update ()
  Updates particle: location and lifetime.
- void **SetSize** (float sz)
  Sets size of the particle.
- void **SetLifeTime** (float l)
  Sets lifetime of the particle.
- void **SetSprite** (SpriteID sp)
  Sets sprite of the particle.
- virtual std::vector< b2Body * > GetPhysBodies ()
  Implements parentclass method and returns empty vector.
- b2Body * **GetBody** ()
  Get mainbody.

Protected Attributes

- float **creationTime_**
- float **size_** = 0.1F
- float **lifeTime_** = 1.0F
- SpriteID **sprite_** = SpriteID::particles_dust

Additional Inherited Members

6.43.1 Detailed Description

Particle class with physics.

6.43.2 Member Function Documentation

6.43.2.1 GetPhysBodies()

```
virtual std::vector< b2Body * > PhysParticle::GetPhysBodies ( )  [inline], [virtual]
```

Implements parentclass method and returns empty vector.

Reimplemented from PhysObject.

6.43.2.2 Render()

```
virtual void PhysParticle::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the particle.

Implements GameObject.

6.43.2.3 Update()

```
virtual void PhysParticle::Update ( )  [inline], [virtual]
```

Updates particle: location and lifetime.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

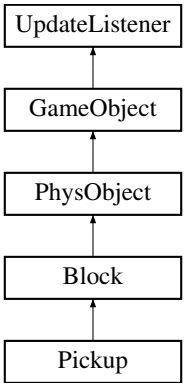
- ParticleEffect.hpp

6.44 Pickup Class Reference

A block that gives a teekkari to the player when broken.

`#include <Pickup.hpp>`

Inheritance diagram for Pickup:



Public Member Functions

- **Pickup** (Game &game, gm::GameObjectType type, float x, float y, float rot)
  Constructor.

Protected Member Functions

- virtual void OnDeath ()
  Creates explosion and points and other related stuff on death.

Additional Inherited Members

6.44.1 Detailed Description

A block that gives a teekkari to the player when broken.

6.44.2 Member Function Documentation

6.44.2.1 OnDeath()

`virtual void Pickup::OnDeath ( ) [inline], [protected], [virtual]`

Creates explosion and points and other related stuff on death.

Reimplemented from Block.

The documentation for this class was generated from the following file:

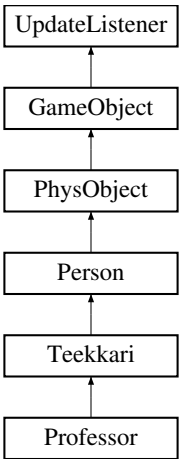
- Pickup.hpp

6.45 Professor Class Reference

Class for the professor.

`#include <Teekkari.hpp>`

Inheritance diagram for Professor:



Public Member Functions

- **Professor** (Game &game, float x, float y, float rot, gm::PersonData data)
- **Professor** (Game &game, float x, float y, float rot)
- virtual void Update ()

  Updates rigidbody.

Protected Member Functions

- virtual void Ability (float x, float y)
- float **GetRealTime** ()

Protected Attributes

- bool **resumed_** = false
- int **updCount_** = 0
- float **abilityStartTime_** = 0
- b2Vec2 **tVelocity_** = {0, 0}
- float **tY_** = 0
- std::vector< ProfessorParticle * > **particles_**

Additional Inherited Members

6.45.1 Detailed Description

Class for the professor.

6.45.2 Member Function Documentation

6.45.2.1 Ability()

```
virtual void Professor::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

6.45.2.2 Update()

```
virtual void Professor::Update ( )  [inline], [virtual]
```

Updates rigidbody.

Reimplemented from Teekkari.

The documentation for this class was generated from the following file:

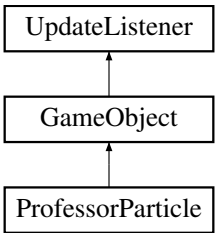
- Teekkari.hpp

6.46 ProfessorParticle Class Reference

Special particle that moves in stopped time.

```
#include <ParticleEffect.hpp>
```

Inheritance diagram for ProfessorParticle:



Public Member Functions

- **ProfessorParticle** (Game &game, float x, float y, float rot)
  Constructor.
- virtual void Render (const RenderSystem &r)
  Renders the particle.
- virtual void Update ()
  Updates particle: location and lifetime.
- void **SetSize** (float sz)
  Sets size of the particle.
- void **SetLifeTime** (float l)
  Sets lifetime of the particle.
- void **SetSprite** (SpriteID sp)
  Sets sprite of the particle.
- virtual std::vector< b2Body * > GetPhysBodies ()
  Implements parentclass method and return empty vector.
- void **SetVelocity** (float x, float y)
  Set velocity of the particle.

Protected Member Functions

- float **GetRealTime** ()

Protected Attributes

- int **upd_** = 0
- float **creationTime_**
- float **size_** = 0.1F
- float **lifeTime_** = 1.0F
- SpriteID **sprite_** = SpriteID::particles_dust
- float **vx** = 0
- float **vy** = 0

6.46.1 Detailed Description

Special particle that moves in stopped time.

6.46.2 Member Function Documentation

6.46.2.1 GetPhysBodies()

```
virtual std::vector< b2Body * > ProfessorParticle::GetPhysBodies ( )  [inline], [virtual]
```

Implements parentclass method and return empty vector.

Reimplemented from GameObject.

6.46.2.2 Render()

```
virtual void ProfessorParticle::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the particle.

Implements GameObject.

6.46.2.3 Update()

```
virtual void ProfessorParticle::Update ( )  [inline], [virtual]
```

Updates particle: location and lifetime.

Reimplemented from UpdateListener.

The documentation for this class was generated from the following file:

- ParticleEffect.hpp

6.47 RenderSystem Class Reference

Framework class for drawing sprites and basic shapes with relative or absolute units.

```
#include <RenderSystem.hpp>
```

Public Member Functions

- **RenderSystem** (sf::RenderWindow &window, ResourceManager &resourceManager)

  Construct a RenderSystem that draws to this Window, and queries sprites from this ResourceManager.
- void RenderSprite (SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot, const Camera &camera, const sf::Color &color=sf::Color(255, 255, 255)) const

  Render a sprite in screen independent coordinates (meters).
- void **RenderSprite** (SpriteID id, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const sf::Color &color=sf::↩ Color(255, 255, 255)) const

  Render a sprite in relative coordinates. This is useful for UI.
- void **RenderSprite** (SpriteID id, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const ui::CropArea &cropArea, const sf::Color &color=sf::Color(255, 255, 255)) const

  Render a sprite in relative coordinates, but restrict drawing to a cropped portion.
- void **RenderRect** (const sf::Color &color, ph::tfloat x, ph::tfloat y, ph::tfloat w, ph::tfloat h, ph::tfloat rot, const Camera &camera) const

  Render a single color rectangle in screen independent coordinates.
- void **RenderRect** (const sf::Color &color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h) const

  Render a single color rectangle in relative coordinates.
- void **RenderRect** (const sf::Color &color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const ui::CropArea &cropArea) const

  Render a single color rectangle in relative coordinates, but restrict drawing to a cropped portion.

- void **RenderText** (const std::string &text, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot, const Camera &camera, const sf::Color &color=ui::textColor, FontID id=ui::defaultFont) const

  Render text in screen independent coordinates. Font size is defined by height and width is dependent on string length.
- void **RenderText** (const std::string &text, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const sf::Color &color=ui::textColor, FontID id=ui::defaultFont, ui::TextAlign textAlign=ui::TextAlign::center) const

  Render text in relative coordinates. Fontsize is defined by height h.
- void **RenderText** (const std::string &text, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const ui::CropArea &cropArea, const sf::Color &color=ui::textColor, FontID id=ui::defaultFont, ui::TextAlign textAlign=ui::Text↩Align::center) const

  Render text in relative coordinates, but restrict drawing to a cropped portion.
- ui::pfloat **MeasureText** (const std::string &text, ui::pfloat h, ui::pfloat::P p=ui::pfloat::P::vw, FontID id=ui↩::defaultFont) const

  Measures the width of string of text at a given height. Returns the units in vw or vh depending on the argument p.
- void **RenderOval** (const sf::Color &color, ph::tfloat x, ph::tfloat y, ph::tfloat w, ph::tfloat h, ph::tfloat rot, const Camera &camera) const

  Render a single color oval in screen independent coordinates.
- void **RenderOval** (const sf::Color &color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h) const

  Render a single color oval in relative coordinates.
- void **RenderOval** (const sf::Color &color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const ui::CropArea &cropArea) const

  Render a single color oval in relative coordinates, but restrict drawing to a cropped portion.
- void **RenderAnimation** (AnimationID id, int frame, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot, const Camera &camera, const sf::Color &color=sf::Color(255, 255, 255)) const

  Render a frame from an animation.
- sf::Vector2f **GetRelativeCoords** (sf::Vector2f coords, const Camera &camera) const

  Get the screen space coordinates of this world position when translated with a Camera.
- sf::Vector2f **GetAbsCoords** (sf::Vector2f coords, const Camera &camera) const

  Get the world position coordinates of this screen space coordinates.
- bool **ContainsCoordinates** (SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot, sf::Vector2f mouseCoords) const

  Check if the given sprite contains the given coordinates.
- sf::Sprite **MakeSprite** (SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot) const

  Make sprite from spriteid and position data.
- bool **IntersectWithSprite** (SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot, sf::Sprite sprite) const

  Check if two sprites intersects.
- bool **CheckGround** (sf::Sprite s) const

  Check if sprite intersects with the ground.

Friends

- class **Application**

6.47.1 Detailed Description

Framework class for drawing sprites and basic shapes with relative or absolute units.

6.47.2 Member Function Documentation

6.47.2.1 RenderSprite()

```
void RenderSystem::RenderSprite (
            SpriteID id,
            ph::tfloat x,
            ph::tfloat y,
            ph::tfloat h,
            ph::tfloat rot,
            const Camera & camera,
            const sf::Color & color = sf::Color(255, 255, 255) ) const
```

Render a sprite in screen independent coordinates (meters).

See also

Physics For the definition of screen independent coordinates

This is useful for game objects. The shape is defined by the sprite itself.

The documentation for this class was generated from the following file:

- RenderSystem.hpp

6.48 ResourceManager Class Reference

Framework class for managing and indexing all resources.

`#include <ResourceManager.hpp>`

Public Member Functions

- **ResourceManager** (const FileManager &)
  Construct a ResourceManager, that loads resources with this FileManager.
- const sf::Font & **GetFont** (FontID)
  Get reference to the Font specified by this FontID.
- const sf::Sprite & **GetSprite** (SpriteID id)
  Get reference to the Sprite specified by this SpriteID.
- const sf::SoundBuffer & **GetSound** (SoundID id)
  Get reference to the Sound specified by this SoundID.
- const sf::Sprite & **GetAnimation** (AnimationID id, int frame)
  Get reference to the sprite specified by this AnimationID and frame.

6.48.1 Detailed Description

Framework class for managing and indexing all resources.

The documentation for this class was generated from the following file:

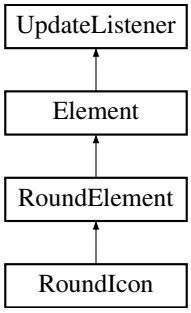
- ResourceManager.hpp

6.49 RoundElement Class Reference

a base calss for elements with a round hit box

`#include <RoundElement.hpp>`

Inheritance diagram for RoundElement:



Public Member Functions

- **RoundElement** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &radius)
- virtual bool isInside (float xw, float yh) const

  checks if the given coordinates are inside the elements and its crop area and the element is visible

Protected Member Functions

- float **getCenterVHFloatX** () const
- float **getCenterVHFloatX** (float rvh) const
- float **getCenterVHFloatY** () const
- float **getCenterVHFloatY** (float rvh) const
- float **distance** (float x1, float y1, float x2, float y2) const

Protected Attributes

- ui::pfloat **r_**

6.49.1 Detailed Description

a base calss for elements with a round hit box

6.49.2 Member Function Documentation

6.49.2.1 isInside()

```
virtual bool RoundElement::isInside (
            float xw,
            float yh ) const  [virtual]
```

checks if the given coordinates are inside the elements and its crop area and the element is visible

Reimplemented from Element.

The documentation for this class was generated from the following file:

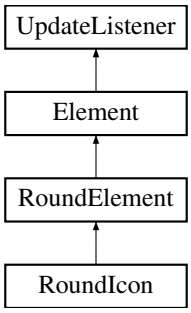
- RoundElement.hpp

6.50 RoundIcon Class Reference

an element for showing icons with a round hit box

`#include <RoundIcon.hpp>`

Inheritance diagram for RoundIcon:



Public Member Functions

- **RoundIcon** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &radius, const SpriteID &icon)
- virtual void Render (const RenderSystem &r)
- void **SetIcon** (const SpriteID &icon)
- SpriteID **GetIcon** ()
- void **Select** ()

  modify the apperance of the element to indicate that it is selected
- void **Unselect** ()

  undo Select()
- void **SetBorderThickness** (const ui::pfloat &)

  set the thickness of the highlight border that is shown when the element is selected

Additional Inherited Members

6.50.1 Detailed Description

an element for showing icons with a round hit box

6.50.2 Member Function Documentation

6.50.2.1 Render()

```
virtual void RoundIcon::Render (
            const RenderSystem & r )  [virtual]
```

Implements Element.

The documentation for this class was generated from the following file:

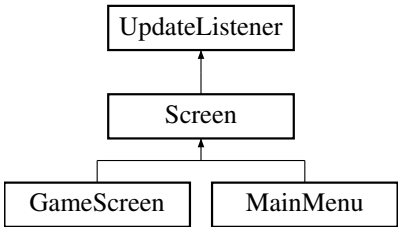
- RoundIcon.hpp

6.51 Screen Class Reference

Base class for screens.

`#include <Screen.hpp>`

Inheritance diagram for Screen:



Public Member Functions

- **Screen** (Application &app)
- virtual void Update ()
- virtual void Render (const RenderSystem &r)
- Application & **GetApplication** () const
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)
- virtual bool OnMouseMove (float xw, float yh)
- virtual bool OnMouseScroll (float delta, float xw, float yh)
- virtual bool OnTextEntered (const sf::Event::TextEvent &)
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)
- virtual bool OnKeyUp (const sf::Event::KeyEvent &)
- void **Confirm** (std::string text, const std::function< void(bool)> callBack)

  add a confirmation message box to screen's message queue
- void **Alert** (std::string text, const std::function< void()> callBack)

  add an alert message box to screen's message queue
- void **Alert** (std::string text)

- void **PlayClickSound** () const

  use audio system to play the UI click sound
- void **DequeueMessage** ()

  remove the oldest message from screen's message queue
- ui::pfloat **calcMessageBoxButtonTop** (const ui::pfloat &messageHeight) const
- ui::pfloat calcMessageBoxButtonLeft (unsigned char buttonNumber, const ui::pfloat &messageWidth) const
- ui::pfloat **calcConfirmTextTop** () const
- ui::pfloat **calcConfirmTextLeft** () const
- ui::pfloat **calcConfirmTextHeight** () const
- ui::pfloat **calcConfirmTextWidth** () const

Protected Member Functions

- template<typename T >
  std::string **getString** (T v) const
- int **parseInt** (std::string s) const
- bool **isEmpty** (std::string s) const
- std::shared_ptr< RoundIcon > generateMessageBoxButton (unsigned char buttonNumber, const std::function< void()> callBack, const SpriteID &sprite, const ui::pfloat &messageHeight, const ui::pfloat &messageWidth)
- std::shared_ptr< TextElement > **generateConfirmText** (const std::string &text)
- void **setFocusedElement** (const std::shared_ptr< Element > &)

Protected Attributes

- const ui::pfloat **messageBoxHeight_** = 15 VH
- const ui::pfloat **messageBoxWidth_** = 30 VW
- const ui::pfloat **messageBoxButtonSize_** = 4 VH
- const ui::pfloat **messageBoxSpacing_** = 1 VH
- Application & **app_**
- std::vector< std::shared_ptr< Element > > **menu_**
- std::queue< std::vector< std::shared_ptr< Element > > > **messages_**
- float **windowWidth_** = 0.0F
- float **windowHeight_** = 0.0F
- std::shared_ptr< Element > **focusedElement_**
- bool **hasFocusedElement_** = false

6.51.1 Detailed Description

Base class for screens.

6.51.2 Member Function Documentation

6.51.2.1 calcMessageBoxButtonLeft()

```
ui::pfloat Screen::calcMessageBoxButtonLeft (
            unsigned char buttonNumber,
            const ui::pfloat & messageWidth ) const
```

button number is the number of the button from right starting from 1.

6.51.2.2 generateMessageBoxButton()

```
std::shared_ptr< RoundIcon > Screen::generateMessageBoxButton (
            unsigned char buttonNumber,
            const std::function< void()> callBack,
            const SpriteID & sprite,
            const ui::pfloat & messageHeight,
            const ui::pfloat & messageWidth )  [protected]
```

button number is the number of the button from right starting from 1.

6.51.2.3 OnKeyDown()

```
virtual bool Screen::OnKeyDown (
            const sf::Event::KeyEvent &  )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.4 OnKeyUp()

```
virtual bool Screen::OnKeyUp (
            const sf::Event::KeyEvent &  )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.5 OnMouseDown()

```
virtual bool Screen::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.6 OnMouseMove()

```
virtual bool Screen::OnMouseMove (
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.7 OnMouseScroll()

```
virtual bool Screen::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.8 OnMouseUp()

```
virtual bool Screen::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.9 OnTextEntered()

```
virtual bool Screen::OnTextEntered (
            const sf::Event::TextEvent &  )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.10 Render()

```
virtual void Screen::Render (
            const RenderSystem & r )  [virtual]
```

Reimplemented from UpdateListener.

6.51.2.11 Update()

```
virtual void Screen::Update ( )  [inline], [virtual]
```

Reimplemented from UpdateListener.

The documentation for this class was generated from the following file:

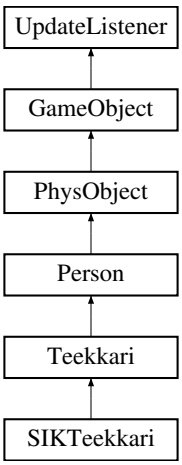
- Screen.hpp

6.52 SIKTeekkari Class Reference

Class for electrical engineering student (SIKteekkari)

```
#include <Teekkari.hpp>
```

Inheritance diagram for SIKTeekkari:



Public Member Functions

- **SIKTeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **SIKTeekkari** (Game &game, float x, float y, float rot)
- virtual void Render (const RenderSystem &r)

  Renders the person.
- virtual void Update ()

  Updates rigidbody.

Protected Member Functions

- virtual void Ability (float x, float y)
- void **ActiveAbility** ()

Protected Attributes

- float **abilityStartTime_** = 0
- bool **used_** = false
- b2Vec2 **lightningPos_** = {0, 0}
- float **lightningH_** = 0
- float **lightningRot_** = 0
- bool **lightning_** = false
- float **lightningStart_** = 0

Additional Inherited Members

6.52.1 Detailed Description

Class for electrical engineering student (SIKteekkari)

6.52.2 Member Function Documentation

6.52.2.1 Ability()

```
virtual void SIKTeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

6.52.2.2 Render()

```
virtual void SIKTeekkari::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the person.

Reimplemented from Person.

6.52.2.3 Update()

```
virtual void SIKTeekkari::Update ( )  [inline], [virtual]
```

Updates rigidbody.

Reimplemented from Teekkari.

The documentation for this class was generated from the following file:

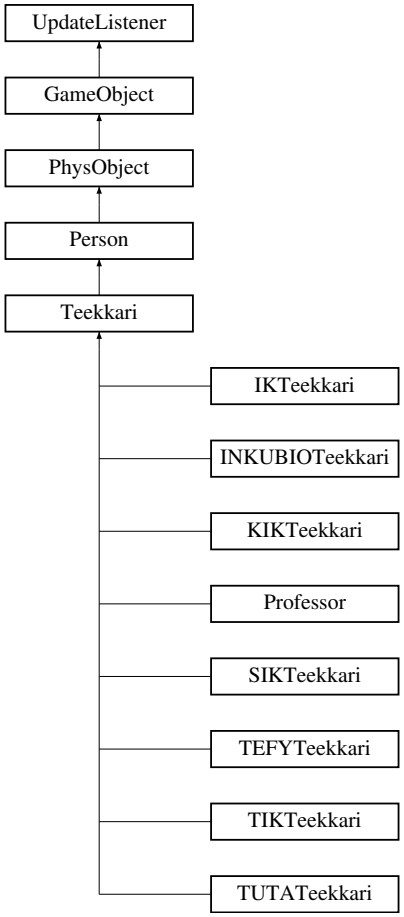
- Teekkari.hpp

6.53 Teekkari Class Reference

Class for the projectiles of the game.

`#include <Teekkari.hpp>`

Inheritance diagram for Teekkari:



Public Member Functions

- **Teekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **Teekkari** (Game &game, gm::GameObjectType type, float x, float y, float rot)
- virtual void Update ()

  Updates rigidbody.
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)

Protected Member Functions

- virtual void OnDeath ()

  This is called just before this object is destroyed from hp.
- virtual void **Ability** (float x, float y)=0

Protected Attributes

- float **creationTime_**
- bool **abilityUsed_** = false
- int **sleepCounter_** = 0

Additional Inherited Members

6.53.1 Detailed Description

Class for the projectiles of the game.

6.53.2 Member Function Documentation

6.53.2.1 OnDeath()

```
virtual void Teekkari::OnDeath ( )  [inline], [protected], [virtual]
```

This is called just before this object is destroyed from hp.

Reimplemented from PhysObject.

6.53.2.2 OnMouseDown()

```
virtual bool Teekkari::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [inline], [virtual]
```

Reimplemented from UpdateListener.

6.53.2.3 Update()

```
virtual void Teekkari::Update ( )  [inline], [virtual]
```

Updates rigidbody.

Reimplemented from Person.

Reimplemented in SIKTeekkari, TEFYTeekkari, TUTATeekkari, KIKTeekkari, and Professor.

The documentation for this class was generated from the following file:

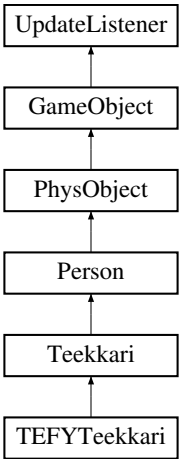
- Teekkari.hpp

6.54 TEFYTeekkari Class Reference

Class for physics student (TEFYTeekkari)

`#include <Teekkari.hpp>`

Inheritance diagram for TEFYTeekkari:



Public Member Functions

- **TEFYTeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **TEFYTeekkari** (Game &game, float x, float y, float rot)
- virtual void Update ()

  Updates rigidbody.
- virtual void Render (const RenderSystem &r)

  Renders the person.

Protected Member Functions

- virtual void Ability (float x, float y)

Protected Attributes

- int **gCounter** = 0
- float **abilityStartTime_** = 0

Additional Inherited Members

6.54.1 Detailed Description

Class for physics student (TEFYTeekkari)

6.54.2 Member Function Documentation

6.54.2.1 Ability()

```
virtual void TEFYTeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

6.54.2.2 Render()

```
virtual void TEFYTeekkari::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the person.

Reimplemented from Person.

6.54.2.3 Update()

```
virtual void TEFYTeekkari::Update ( )  [inline], [virtual]
```

Updates rigidbody.

Reimplemented from Teekkari.

The documentation for this class was generated from the following file:

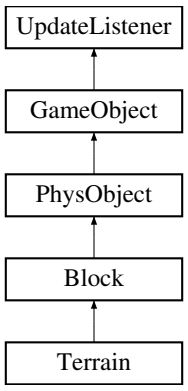
- Teekkari.hpp

6.55 Terrain Class Reference

An inmovable, indestructible block.

`#include <Terrain.hpp>`

Inheritance diagram for Terrain:



Public Member Functions

- **Terrain** (Game &game, float x, float y, float rot)

  Constructor.
- virtual float GetMass () const

  Get mass.
- virtual void DealDamage (float damage)

  Empty method removes ability to take damage.

Additional Inherited Members

6.55.1 Detailed Description

An inmovable, indestructible block.

6.55.2 Member Function Documentation

6.55.2.1 DealDamage()

```
virtual void Terrain::DealDamage (
            float damage )  [inline], [virtual]
```

Empty method removes ability to take damage.

Reimplemented from PhysObject.

6.55.2.2 GetMass()

```
virtual float Terrain::GetMass ( ) const  [inline], [virtual]
```

Get mass.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

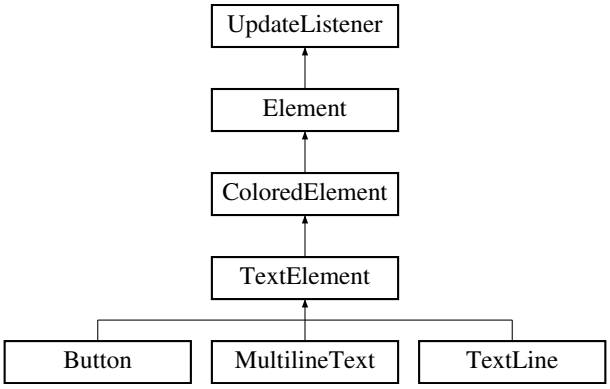
- Terrain.hpp

6.56 TextElement Class Reference

a base class for elements containing text

`#include <TextElement.hpp>`

Inheritance diagram for TextElement:



Public Member Functions

- **TextElement** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width)
- virtual void **SetText** (const std::string &s)
- void **SetTextColor** (const sf::Color &c)
- void **SetTextColor** ()
- void **SetFont** (FontID f)
- void **SetTextAlign** (const ui::TextAlign &a)
- void **SetRelativeFontSize** (const ui::pfloat &s)
- void **SetAbsoluteFontSize** (float s)
- virtual void Render (const RenderSystem &)
- ui::pfloat **GetFontSize** ()

Protected Attributes

- std::string **text_** = ""
- sf::Color **textColor_** = ui::textColor
- FontID **font_** = ui::defaultFont
- ui::TextAlign **align_** = ui::TextAlign::left

Additional Inherited Members

6.56.1 Detailed Description

a base class for elements containing text

6.56.2 Member Function Documentation

6.56.2.1 Render()

```
virtual void TextElement::Render (
            const RenderSystem &  )  [virtual]
```

Reimplemented from ColoredElement.

The documentation for this class was generated from the following file:

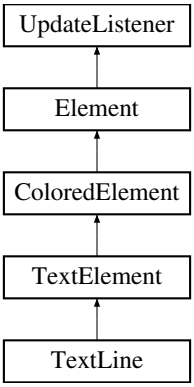
- TextElement.hpp

6.57 TextLine Class Reference

an element for a single text line

`#include <TextLine.hpp>`

Inheritance diagram for TextLine:



Public Member Functions

- **TextLine** (const ui::pfloat &top, const ui::pfloat &left, const ui::pfloat &height, const ui::pfloat &width, const std::string &text)
- virtual void Render (const RenderSystem &)

Additional Inherited Members

6.57.1 Detailed Description

an element for a single text line

6.57.2 Member Function Documentation

6.57.2.1 Render()

```
virtual void TextLine::Render (
            const RenderSystem &  )  [virtual]
```

Reimplemented from TextElement.

The documentation for this class was generated from the following file:

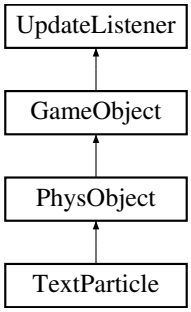
- TextLine.hpp

6.58 TextParticle Class Reference

Particle class for texts.

`#include <ParticleEffect.hpp>`

Inheritance diagram for TextParticle:



Public Member Functions

- **TextParticle** (Game &game, float x, float y, float rot)
  Constructor.
- virtual void Render (const RenderSystem &r)
  Renders the particle.
- virtual void Update ()
  Updates particle: location and lifetime.
- void **SetSize** (float sz)
  Sets size of the particle.
- void **SetLifeTime** (float l)
  Sets lifetime of the particle.
- void **SetText** (std::string text)
  Sets text of the particle.
- void **SetColor** (sf::Color color)
  Sets color of the particle.
- virtual std::vector< b2Body * > GetPhysBodies ()
  Implements parentclass method and return empty vector.

Protected Attributes

- float **creationTime_**
- float **size_** = 0.1F
- float **lifeTime_** = 1.0F
- std::string **text_** = ""
- sf::Color **color_** = {255, 106, 0, 255}

Additional Inherited Members

6.58.1 Detailed Description

Particle class for texts.

6.58.2 Member Function Documentation

6.58.2.1 GetPhysBodies()

```
virtual std::vector< b2Body * > TextParticle::GetPhysBodies ( )  [inline], [virtual]
```

Implements parentclass method and return empty vector.

Reimplemented from PhysObject.

6.58.2.2 Render()

```
virtual void TextParticle::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the particle.

Implements GameObject.

6.58.2.3 Update()

```
virtual void TextParticle::Update ( )  [inline], [virtual]
```

Updates particle: location and lifetime.

Reimplemented from PhysObject.

The documentation for this class was generated from the following file:

- ParticleEffect.hpp

6.59 ph::tfloat Struct Reference

A struct to help with interpolating.

`#include <Physics.hpp>`

Public Member Functions

- **tfloat** (const float &f)
- **operator float** () const
- tfloat & **operator=** (const float &f)
- tfloat & **operator*=** (const float &f)
- tfloat & **operator/=** (const float &f)
- tfloat & **operator+=** (const float &f)
- tfloat & **operator-=** (const float &f)
- float **Lerp** (float t) const
- void **Record** ()

Public Attributes

- float **f0**
- float **f1**

6.59.1 Detailed Description

A struct to help with interpolating.

A tfloat is simply a float that keeps track of its last value Treat these like any normal float. In fact, you can freely assign floats to tfloats, and tfloats to floats

This is used by the RenderSystem to be able to interpolate positions, sizes, anything. Using tfloats simply gives a clean way to incorporate interpolation to the whole physics system, without actually doing it IN the physics system. Call Record at the start of an Update

The documentation for this struct was generated from the following file:

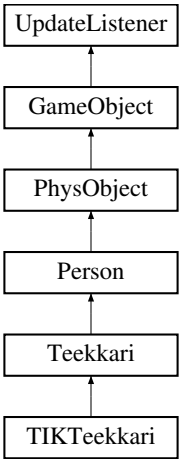
- Physics.hpp

6.60 TIKTeekkari Class Reference

Class for computer science student (TIKteekkari)

`#include <Teekkari.hpp>`

Inheritance diagram for TIKTeekkari:



Public Member Functions

- **TIKTeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **TIKTeekkari** (Game &game, float x, float y, float rot)

Protected Member Functions

- virtual void Ability (float x, float y)

Additional Inherited Members

6.60.1 Detailed Description

Class for computer science student (TIKteekkari)

6.60.2 Member Function Documentation

6.60.2.1 Ability()

```
virtual void TIKTeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

The documentation for this class was generated from the following file:

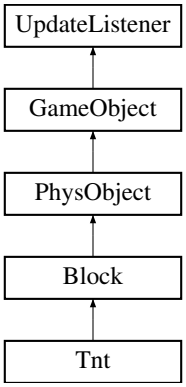
- Teekkari.hpp

6.61 Tnt Class Reference

A TNT block that explodes.

`#include <Tnt.hpp>`

Inheritance diagram for Tnt:



Public Member Functions

- **Tnt** (Game &game, float x, float y, float rot)
  Constructor.

Protected Member Functions

- virtual void OnDeath ()
  Creates explosion and damege on death.

Additional Inherited Members

6.61.1 Detailed Description

A TNT block that explodes.

6.61.2 Member Function Documentation

6.61.2.1 OnDeath()

`virtual void Tnt::OnDeath ( )  [inline], [protected], [virtual]`

Creates explosion and damege on death.

Reimplemented from Block.

The documentation for this class was generated from the following file:

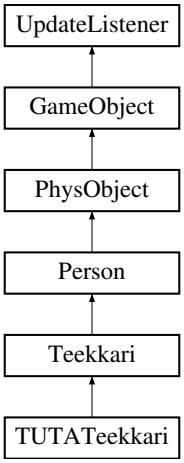
- Tnt.hpp

6.62 TUTATeekkari Class Reference

Class for industrial engineering student (TUTAteekkari)

`#include <Teekkari.hpp>`

Inheritance diagram for TUTATeekkari:



Public Member Functions

- **TUTATeekkari** (Game &game, float x, float y, float rot, gm::PersonData data)
- **TUTATeekkari** (Game &game, float x, float y, float rot)
- virtual void Update ()

  Updates rigidbody.
- virtual void Render (const RenderSystem &r)

  Renders the person.

Protected Member Functions

- virtual void Ability (float x, float y)

Protected Attributes

- int **whooshCounter** = 0
- float **abilityStartTime_** = 0

Additional Inherited Members

6.62.1 Detailed Description

Class for industrial engineering student (TUTAteekkari)

6.62.2 Member Function Documentation

6.62.2.1 Ability()

```
virtual void TUTATeekkari::Ability (
            float x,
            float y )  [inline], [protected], [virtual]
```

Implements Teekkari.

6.62.2.2 Render()

```
virtual void TUTATeekkari::Render (
            const RenderSystem & r )  [inline], [virtual]
```

Renders the person.

Reimplemented from Person.

6.62.2.3 Update()

```
virtual void TUTATeekkari::Update ( )  [inline], [virtual]
```

Updates rigidbody.

Reimplemented from Teekkari.

The documentation for this class was generated from the following file:

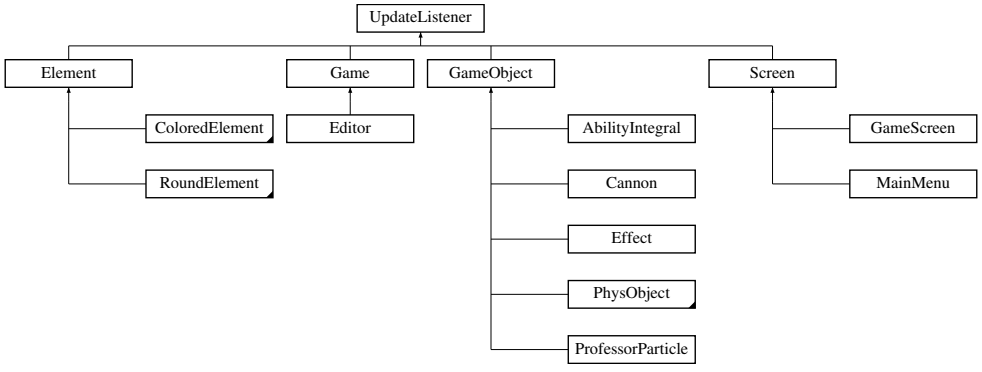
- Teekkari.hpp

6.63 UpdateListener Class Reference

A base class of an update handler.

`#include <UpdateListener.hpp>`

Inheritance diagram for UpdateListener:



Public Member Functions

- virtual void Update ()
- virtual void Render (const RenderSystem &)
- virtual bool OnKeyDown (const sf::Event::KeyEvent &)
- virtual bool **OnKeyUp** (const sf::Event::KeyEvent &)
- virtual bool OnMouseDown (const sf::Mouse::Button &button, float xw, float yh)
- virtual bool OnMouseUp (const sf::Mouse::Button &button, float xw, float yh)
- virtual bool OnMouseMove (float xw, float yh)
- virtual bool OnMouseScroll (float delta, float xw, float yh)
- virtual bool **OnTextEntered** (const sf::Event::TextEvent &)

6.63.1 Detailed Description

A base class of an update handler.

A class inheriting UpdateListener can receive updates

6.63.2 Member Function Documentation

6.63.2.1 OnKeyDown()

```
virtual bool UpdateListener::OnKeyDown (
            const sf::Event::KeyEvent &  )  [inline], [virtual]
```

Reimplemented in Editor.

6.63.2.2 OnMouseDown()

```
virtual bool UpdateListener::OnMouseDown (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [inline], [virtual]
```

Reimplemented in Editor, Game, Element, MessageBox, and Cannon.

6.63.2.3 OnMouseMove()

```
virtual bool UpdateListener::OnMouseMove (
            float xw,
            float yh )  [inline], [virtual]
```

Reimplemented in Cannon, Editor, and Game.

6.63.2.4 OnMouseScroll()

```
virtual bool UpdateListener::OnMouseScroll (
            float delta,
            float xw,
            float yh )  [inline], [virtual]
```

Reimplemented in Game.

6.63.2.5 OnMouseUp()

```
virtual bool UpdateListener::OnMouseUp (
            const sf::Mouse::Button & button,
            float xw,
            float yh )  [inline], [virtual]
```

Reimplemented in Editor, Game, and Cannon.

6.63.2.6 Render()

```
virtual void UpdateListener::Render (
            const RenderSystem &  )  [inline], [virtual]
```

Reimplemented in GameObject, Block, Cannon, Effect, Game, Ground, PhysParticle, TextParticle, ProfessorParticle, Person, AbilityCow, AbilityWrench, AbilityIntegral, SIKTeekkari, TEFYTeekkari, and TUTATeekkari.

6.63.2.7 Update()

```
virtual void UpdateListener::Update ( )  [inline], [virtual]
```

Reimplemented in Cannon, Effect, Game, Ground, PhysParticle, TextParticle, ProfessorParticle, Person, PhysObject, Teekkari, AbilityCow, AbilityWrench, SIKTeekkari, TEFYTeekkari, TUTATeekkari, KIKTeekkari, and Professor.

The documentation for this class was generated from the following file:

- UpdateListener.hpp

Chapter 7

File Documentation

7.1 Application.hpp

```
1 #ifndef APPLICATION_HPP
2 #define APPLICATION_HPP
3
4 #include <memory>
5 #include <screens/Screen.hpp>
6 #include <SFML/Graphics.hpp>
7 #include <framework/FileManager.hpp>
8 #include <framework/ResourceManager.hpp>
9 #include <framework/RenderSystem.hpp>
10 #include <framework/AudioSystem.hpp>
11
12
13 class Screen;
14
16
17 class Application {
18 public:
19
21     Application();
22
24     bool Loop();
25
27     void TransitionTo(std::unique_ptr<Screen>);
28
30     void Resize(unsigned int width, unsigned int height);
31
33     void Fullscreen();
34
36     void Exit();
37
38
39     // These are here in case someone absolutely needs them
40
42     float GetAspectRatio() const;
43
45     float GetWindowWidth() const;
46
48     float GetWindowHeight() const;
49
51     bool IsFullScreen() const;
52
54     AudioSystem& GetAudioSystem();
55
57     const FileManager& GetFileManager() const;
58
60     const RenderSystem& GetRenderSystem() const;
61
63     const ResourceManager& GetResourceManager() const;
64
65
66
67 private:
68
69     //Time stuff
70     sf::Clock clock;
71     float accumulatedTime = 0;
72     bool isFullScreen_ = true;
73
```

```
74     sf::RenderWindow window_;
75     std::unique_ptr<Screen> activeScreen_;
76
77     FileManager fileManager_;
78     AudioSystem audioSystem_;
79     RenderSystem renderSystem_;
80     ResourceManager resourceManager_;
81
82     void UpdateView();
83
84 };
85
86
87
88 #endif
```

7.2 deprecated.hpp

```
1 #ifndef DEPRECATED_HPP
2 #define DEPRECATED_HPP
3 #if defined(__GNUC__) || defined(__clang__)
4 #define DEPRECATED __attribute__((deprecated))
5 #elif defined(_MSC_VER)
6 #define DEPRECATED __declspec(deprecated)
7 #else
8 #define DEPRECATED
9 #endif
10 #endif
```

7.3 AudioSystem.hpp

```
1 #ifndef AUDIO_SYSTEM_HPP
2 #define AUDIO_SYSTEM_HPP
3
4 #include <queue>
5 #include <SFML/Audio.hpp>
6 #include <framework/ResourceManager.hpp>
7
9 class AudioSystem {
10 public:
11
13     AudioSystem(ResourceManager& resourceManager) : resourceManager_(resourceManager) {}
14
16     void PlaySound(SoundID id, float volume = 1.0F);
17
19     void SetGlobalVolume(float volume);
20
21 private:
22
23     ResourceManager& resourceManager_;
24     static const int queueSize = 18;
25     float globalVolume_ = 1.0F;
26
27     std::queue<sf::Sound> soundQueue_;
28
30     void Cleanup();
31
32 };
33
34 #endif
```

7.4 FileManager.hpp

```
1 #ifndef FILE_MANAGER_HPP
2 #define FILE_MANAGER_HPP
3
4 #include <vector>
5 #include <string>
6 #include <fstream>
7 #include <SFML/Graphics.hpp>
8 #include <gameplay/Level.hpp>
9 #include <SFML/Audio/SoundBuffer.hpp>
10
11
12
```

```
13
15 class FileManager {
16 public:
17
19     bool LoadTexture(sf::Texture& texture, const std::string& path) const;
20
22     bool LoadAudio(sf::SoundBuffer& soundBuffer, const std::string& path) const;
23
25     bool LoadFont(sf::Font& font, const std::string& path) const;
26
27
29     std::vector<Level> ListLevels() const;
30
32     std::vector<Level> ListEndless() const;
33
35     bool SaveLevel(Level& level) const;
36
38     void DeleteLevel(const Level& level) const;
39
40 private:
41
42     const std::string levelPath = "data/levels/";
43     const std::string endlessPath = "data/levels/endless/";
44
46     std::string GenerateFilepath(const std::string folder) const;
47
49     bool LoadLevel(Level& level, const std::string& path) const;
50
52     bool SaveLevel(const Level& level, const std::string& path) const;
53
55     void PrintGameObjectData(std::ofstream& file, const gm::GameObjectData& data) const;
56
58     void PrintHighScores(std::ofstream& file, const std::pair<std::string, int>& score) const;
59
61     void PrintStartingTeekkaris(std::ofstream& file, const gm::GameObjectType& object) const;
62
64     std::vector<std::string> ListLevelPaths(std::string folder) const;
65
67     std::vector<Level> LoadLevels(std::string path) const;
68
69
70 };
71
72
73 #endif
```

7.5 RandomGen.hpp

```
1 #ifndef RANDOM_GEN_HPP
2 #define RANDOM_GEN_HPP
3
4 #include <random>
5 #include <limits>
6
8 namespace rng {
9
10 inline std::mt19937 engine;
11
13 inline void InitializeRng() {
14     std::random_device dev;
15     rng::engine = std::mt19937{dev()};
16 }
17
19 inline unsigned int RandomInt(unsigned int min = 0, int max = std::numeric_limits<unsigned int>::max()) {
20     std::uniform_int_distribution<std::default_random_engine::result_type> d(min, max);
21     return d(engine);
22 }
23
25 inline float RandomF() {
26     std::uniform_real_distribution<float> f(0.0F, 1.0F);
27     return f(engine);
28 }
29
30
31 }
32
33
34 #endif
```

7.6 RenderSystem.hpp

```
1 #ifndef RENDER_SYSTEM_HPP
2 #define RENDER_SYSTEM_HPP
3
4 #include <string>
5 #include <gameplay/Camera.hpp>
6 #include <ui/UIConstants.hpp>
7 #include <gameplay/Physics.hpp>
8 #include <SFML/Graphics.hpp>
9 #include <framework/ResourceManager.hpp>
10
12 class RenderSystem {
13 public:
14
16     RenderSystem(sf::RenderWindow& window, ResourceManager& resourceManager) :
       resourceManager_(resourceManager), window_(window) {}
17
19
23     void RenderSprite(SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot,
24     const Camera& camera, const sf::Color& color = sf::Color(255, 255, 255)) const;
25
27     void RenderSprite(SpriteID id, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h,
28     const sf::Color& color = sf::Color(255, 255, 255)) const;
29
31     void RenderSprite(SpriteID id, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h,
32     const ui::CropArea& cropArea, const sf::Color& color = sf::Color(255, 255, 255)) const;
33
35     void RenderRect(const sf::Color& color, ph::tfloat x, ph::tfloat y, ph::tfloat w, ph::tfloat h,
       ph::tfloat rot, const Camera& camera) const;
36
38     void RenderRect(const sf::Color& color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h)
       const;
39
41     void RenderRect(const sf::Color& color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const
       ui::CropArea& cropArea) const;
42
44     void RenderText(const std::string& text, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot,
45     const Camera& camera, const sf::Color& color = ui::textColor, FontID id = ui::defaultFont) const;
46
48     void RenderText(const std::string& text, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h,
49     const sf::Color& color = ui::textColor, FontID id = ui::defaultFont, ui::TextAlign textAlign =
       ui::TextAlign::center) const;
50
52     void RenderText(const std::string& text, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h,
53     const ui::CropArea& cropArea, const sf::Color& color = ui::textColor, FontID id = ui::defaultFont,
       ui::TextAlign textAlign = ui::TextAlign::center) const;
54
56     ui::pfloat MeasureText(const std::string& text, ui::pfloat h, ui::pfloat::P p = ui::pfloat::P::vw,
       FontID id = ui::defaultFont) const;
57
59     void RenderOval(const sf::Color& color, ph::tfloat x, ph::tfloat y, ph::tfloat w, ph::tfloat h,
       ph::tfloat rot, const Camera& camera) const;
60
62     void RenderOval(const sf::Color& color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h)
       const;
63
65     void RenderOval(const sf::Color& color, ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const
       ui::CropArea& cropArea) const;
66
68     void RenderAnimation(AnimationID id, int frame, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat
       rot,
69     const Camera& camera, const sf::Color& color = sf::Color(255, 255, 255)) const;
70
72     sf::Vector2f GetRelativeCoords(sf::Vector2f coords, const Camera& camera) const;
73
75     sf::Vector2f GetAbsCoords(sf::Vector2f coords, const Camera& camera) const;
76
78     bool ContainsCoordinates(SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot,
       sf::Vector2f mouseCoords) const;
79
81     sf::Sprite MakeSprite(SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot) const;
82
84     bool IntersectWithSprite(SpriteID id, ph::tfloat x, ph::tfloat y, ph::tfloat h, ph::tfloat rot,
       sf::Sprite sprite) const;
85
87     bool CheckGround(sf::Sprite s) const;
88
89 private:
90
91     //Allow Application to set these values
92     friend class Application;
93
95     float ALPHA;
96
98     float WW;
99
```

```
101     float HH;
102
103
104
105
106     sf::RenderWindow& window_;
107     ResourceManager& resourceManager_;
108
109
110
112     void CameraDraw(const sf::Drawable& shape, const Camera& camera) const;
113
115     void RenderRelative(sf::Shape& shape, const sf::Color& color,
116     ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h) const;
117
119     void RenderRelativeCrop(sf::Shape& shape, const sf::Color& color,
120     ui::pfloat x, ui::pfloat y, ui::pfloat w, ui::pfloat h, const ui::CropArea& cropArea) const;
121
123     void RenderAbs(sf::Shape& shape, const sf::Color& color,
124     ph::tfloat x, ph::tfloat y, ph::tfloat w, ph::tfloat h, ph::tfloat rot, const Camera& camera) const;
125
126 };
127
128 #endif
```

7.7 ResourceManager.hpp

```
1 #ifndef RESOURCE_MANAGER_HPP
2 #define RESOURCE_MANAGER_HPP
3
4 #include <vector>
5 #include <utility>
6 #include <unordered_map>
7 #include <SFML/Graphics.hpp>
8 #include <framework/Resources.hpp>
9 #include <framework/FileManager.hpp>
10 #include <SFML/Audio/SoundBuffer.hpp>
11
13 class ResourceManager {
14 public:
15
17     ResourceManager(const FileManager&);
18
20     const sf::Font& GetFont(FontID);
21
23     const sf::Sprite& GetSprite(SpriteID id);
24
26     const sf::SoundBuffer& GetSound(SoundID id);
27
29     const sf::Sprite& GetAnimation(AnimationID id, int frame);
30
31 private:
32
34     std::unordered_map<FontID, sf::Font> fonts_;
35
37     std::unordered_map<SoundID, sf::SoundBuffer> audio_;
38
40     std::unordered_map<SpriteID, sf::Sprite> sprites_;
41
43     std::unordered_map<int, sf::Texture> textures_;
44
46     std::unordered_map<AnimationID, std::vector<sf::Sprite» animations_;
47
48
49     const FileManager& fileManager_;
50
52     sf::Texture missingTexture_;
53
55     struct SpriteMapping {
56         SpriteID spriteID;
57         int textureID;
58         sf::IntRect rect;
59     };
60
61     struct AnimationMapping {
62         AnimationID animationID;
63         int textureID;
64         std::pair<int, int> spriteSize;
65         sf::IntRect areaRect;
66     };
67
69
71     static const std::pair<FontID, std::string> fontPaths_[];
```

```
72
74     static const std::pair<SoundID, std::string> audioPaths_[];
75
77     static const std::pair<int, std::string> texturePaths_[];
78
80     static const SpriteMapping spriteMaps_[];
81
83     static const AnimationMapping animationMaps_[];
84
85 };
86
87 #endif
```

7.8 Resources.hpp

```
1 #ifndef RESOURCES_HPP
2 #define RESOURCES_HPP
3
5 enum SpriteID {
6
7     //UI sprites
8     ui_button,
9     ui_button_pause,
10     ui_button_restart,
11     ui_button_exit,
12     ui_button_resume,
13     ui_button_cancel,
14     ui_button_ok,
15     ui_button_save,
16     ui_button_right,
17     ui_button_left,
18     ui_star,
19     ui_missing_star,
20
21
22     //Level backgrounds
23     background_field,
24
25
26     //Background objects
27     terrain,
28     bg_tree1,
29     bg_tree2,
30     bg_lamp_pole,
31     bg_bench,
32     bg_person1,
33     bg_person2,
34     bg_person3,
35
36
37     //Blocks
38     wood_block1x1,
39     metal_block1x1,
40     glass_block1x1,
41     plastic_block1x1,
42     rubber_block1x1,
43     concrete_block1x1,
44
45     wood_block2x1,
46     metal_block2x1,
47     glass_block2x1,
48     plastic_block2x1,
49     rubber_block2x1,
50     concrete_block2x1,
51
52     wood_block2x2,
53     metal_block2x2,
54     glass_block2x2,
55     plastic_block2x2,
56     rubber_block2x2,
57     concrete_block2x2,
58
59     wood_ball,
60     metal_ball,
61     glass_ball,
62     plastic_ball,
63     rubber_ball,
64     concrete_ball,
65
66     /*
67     wood_blockTri,
68     metal_blockTri,
69     glass_blockTri,
```

```
70      plastic_blockTri,
71      rubber_blockTri,
72      concrete_blockTri,*/
73
74      wood_plank,
75      metal_plank,
76      glass_plank,
77      plastic_plank,
78      rubber_plank,
79      concrete_plank,
80
81      wood_thickplank,
82      metal_thickplank,
83      glass_thickplank,
84      plastic_thickplank,
85      rubber_thickplank,
86      concrete_thickplank,
87
88      //Block crack overlays
89      crack1x1,
90      crack2x1,
91      crack2x2,
92      crack_ball,
93      crack_plank,
94      crack_thickplank,
95
96      crack1x1_b,
97      crack2x1_b,
98      crack2x2_b,
99      crack_ball_b,
100      crack_plank_b,
101      crack_thickplank_b,
102
103      //Props
104      beer,
105      beer_can,
106
107      sofa3x1,
108      tnt,
109
110      //Guilds
111
112      guild_ik,
113      guild_sik,
114      guild_tefy,
115      guild_tuta,
116      guild_tik,
117      guild_inkubio,
118      guild_kik,
119      guild_professor,
120
121
122      //Cannon
123      cannon_base,
124      cannon_head,
125
126
127      //Teekkari parts
128
129      //IK
130      arm_blue,
131      armb_blue,
132      torso_blue,
133      leg_blue,
134
135      //TEFY
136      arm_lwhite,
137      armb_lwhite,
138      torso_lwhite,
139      leg_lwhite,
140
141      //TIK
142      arm_black,
143      armb_black,
144      torso_black,
145      leg_black,
146
147      //INKUBIO
148      arm_brown,
149      armb_brown,
150      torso_brown,
151      leg_brown,
152
153      //TUTA
154      arm_rainbow,
155      armb_rainbow,
156      torso_rainbow,
```

```
157     leg_rainbow,
158
159     //SIK
160     arm_white,
161     armb_white,
162     torso_white,
163     leg_white,
164
165     //KIK
166     arm_pink,
167     armb_pink,
168     torso_pink,
169     leg_pink,
170
171     //Professor
172     professor_head,
173     professor_torso,
174     professor_arm,
175     professor_leg,
176
177
178     //Teekkari heads
179     teekkari_head1,
180     teekkari_head2,
181     teekkari_head3,
182     teekkari_head4,
183     teekkari_head5,
184     teekkari_head6,
185     teekkari_head7,
186     teekkari_head8,
187     teekkari_head9,
188     teekkari_head10,
189
190     teekkari_head1s,
191     teekkari_head2s,
192     teekkari_head3s,
193     teekkari_head4s,
194     teekkari_head5s,
195     teekkari_head6s,
196     teekkari_head7s,
197     teekkari_head8s,
198     teekkari_head9s,
199     teekkari_head10s,
200
201     //Fuksi parts
202
203     //IK
204     fuksi_arm_blue,
205     fuksi_armb_blue,
206     fuksi_torso_blue,
207     fuksi_leg_blue,
208
209     //TEFY
210     fuksi_arm_lwhite,
211     fuksi_armb_lwhite,
212     fuksi_torso_lwhite,
213     fuksi_leg_lwhite,
214
215     //TIK
216     fuksi_arm_black,
217     fuksi_armb_black,
218     fuksi_torso_black,
219     fuksi_leg_black,
220
221     //INKUBIO
222     fuksi_arm_brown,
223     fuksi_armb_brown,
224     fuksi_torso_brown,
225     fuksi_leg_brown,
226
227     //TUTA
228     fuksi_arm_rainbow,
229     fuksi_armb_rainbow,
230     fuksi_torso_rainbow,
231     fuksi_leg_rainbow,
232
233     //SIK
234     fuksi_arm_white,
235     fuksi_armb_white,
236     fuksi_torso_white,
237     fuksi_leg_white,
238
239     //KIK
240     fuksi_arm_pink,
241     fuksi_armb_pink,
242     fuksi_torso_pink,
243     fuksi_leg_pink,
```

```
244
245
246     //Fuksi heads
247     fuksi_head1,
248     fuksi_head2,
249     fuksi_head3,
250     fuksi_head4,
251     fuksi_head5,
252     fuksi_head6,
253     fuksi_head7,
254     fuksi_head8,
255
256
257     //Teekkari abilities
258     gravity_symbols,
259     cow,
260     lightning_strike,
261     wrench,
262     math_cloud,
263     integral_sign,
264
265     //Block break particles
266     particles_dust, //Generic ground hit
267     particles_wood,
268     particles_metal,
269     particles_glass,
270     particles_plastic,
271     particles_rubber,
272     particles_concrete,
273
274     //Prop particles
275     bottle_particle,
276     can_particle,
277
278     //Object collisions
279     hit_sparks1,
280     hit_sparks2,
281
282     //Person collisions
283     hit_stars,
284
285     //Screen filters (sprite applied after all)
286     filter_timefreeze
287
288
289 };
290
292 enum AnimationID {
293
294     //Explosions
295     explosion,
296     cannon_explosion,
297
298     //Person despawning
299     particles_poof,
300
301
302     //Teekkari abilities
303     gravity_spiral,
304     matrix_bug,
305     hand_whirl,
306     thunder_sparks,
307     lightning
308
309
310 };
311
313 enum SoundID {
314
315
316     //UI sounds
317     ui_click,
318
319
320     //Block sounds
321     wood_hit,
322     metal_hit,
323     glass_hit,
324     plastic_hit,
325     rubber_hit,
326     concrete_hit,
327
328     wood_crack,
329     metal_crack,
330     glass_crack,
331     plastic_crack,
332     rubber_crack,
```

```
333     concrete_crack,
334
335
336     //Prop sounds
337     bottle_hit,
338     can_hit,
339     bottle_break,
340     can_break,
341
342     sofa_spring,
343     tnt_explode1,
344     tnt_explode2,
345
346
347     //Cannon sounds
348     cannon_load,
349     cannon_shot,
350
351
352     //Teekkari sounds
353     poof,
354     thud1,
355     thud2,
356     thud3,
357
358     smack1,
359     smack2,
360     smack3,
361
362     grunt1,
363     grunt2,
364     grunt3,
365     grunt4,
366
367     teekkari_death1,
368     teekkari_death2,
369     teekkari_death3,
370     teekkari_death4,
371
372     teekkari_recruit,
373
374
375     //Fuksi sounds
376     fuksi_cry1,
377     fuksi_cry2,
378     fuksi_cry3,
379     fuksi_cry4,
380
381     fuksi_death1,
382     fuksi_death2,
383     fuksi_death3,
384     fuksi_death4,
385
386
387     //Teekkari abilities
388     gravity_shiftup,
389     gravity_shiftdown,
390     glitch_sound,
391     hand_whoosh,
392     wrench_swish,
393     cow_moo,
394     cow_death,
395     thunder_static,
396     thunder_strike,
397     professor_oneliner1,
398     professor_oneliner2,
399     professor_oneliner3,
400     professor_oneliner4,
401     professor_oneliner5,
402     professor_oneliner6,
403     professor_oneliner7,
404     professor_oneliner8,
405     integral_destruction
406
407 };
408
410 enum FontID { source_serif, consolas };
411
412 #endif
```

7.9 Beer.hpp

```
1 #ifndef GAME_BEER_HPP
```

```
2 #define GAME_BEER_HPP
3
4 #include <gameplay/Block.hpp>
5 #include <gameplay/PhysObject.hpp>
6 #include <gameplay/GameObjectTypes.hpp>
7 #include <framework/RenderSystem.hpp>
8 #include <framework/RandomGen.hpp>
9 #include <gameplay/ParticleEffect.hpp>
10 #include <box2d/b2_circle_shape.h>
11 #include <box2d/b2_fixture.h>
12 #include <box2d/b2_world.h>
13 #include <box2d/b2_body.h>
14 #include <box2d/b2_api.h>
15 #include <unordered_map>
16
18 class Beer : public Block {
19 public:
21     Beer(Game& game, gm::GameObjectType type, float x, float y, float rot) : Block(game, type, x, y, rot)
      {}
22
23 protected:
25     virtual void OnDeath() {
26
27         game_.GetAudioSystem().PlaySound(materialData_.breakSound);
28         game_.AddPoints(objectType_ == gm::GameObjectType::prop_beer ? -4000 : -2000);
29
30         //Spawn some particles
31
32         if(objectType_ == gm::GameObjectType::prop_beer) {
33             for(int i = 0; i < 10; i++) {
34                 //Random point inside circle
35                 float a = 2.0F * ph::pi * rng::RandomF();
36                 float u = rng::RandomF() + rng::RandomF();
37                 float r = (u > 1) ? 2 - u : u;
38                 float x = shapeData_.height * r * cosf(a);
39                 float y = shapeData_.height * r * sinf(a);
40                 int id = game_.AddObject(std::make_unique<PhysParticle>(game_, x_ + x, y_ + y,
    ph::angToRot(a)));
41                 PhysParticle& p = (PhysParticle&)game_.GetObject(id);
42
43                 p.SetSize(0.25F);
44                 p.SetSprite(SpriteID::bottle_particle);
45                 p.Angular(rng::RandomInt(0, 1) ? rng::RandomF() * 0.4F : -rng::RandomF() * 0.4F);
46                 p.Explosion({x_, y_}, 30.0F);
47             }
48         }
49
50         else {
51             float a = 2.0F * ph::pi * rng::RandomF();
52             int id = game_.AddObject(std::make_unique<PhysParticle>(game_, x_, y_, rot_));
53             PhysParticle& p = (PhysParticle&)game_.GetObject(id);
54
55             p.SetSize(0.25F);
56             p.SetSprite(SpriteID::can_particle);
57             p.Angular(rng::RandomInt(0, 1) ? rng::RandomF() * 0.4F : -rng::RandomF() * 0.4F);
58             p.Impulse({0, 10.0F});
59         }
60
61         int id = game_.AddObject(std::make_unique<TextParticle>(game_, x_+0.5F, y_, 0.0F));
62         TextParticle& textP = (TextParticle&)game_.GetObject(id);
63         textP.SetSize(1.0F);
64         textP.SetColor(sf::Color(0, 0, 0, 255));
65         textP.SetText(objectType_ == gm::GameObjectType::prop_beer ? "-4000" : "-2000");
66
67
68     }
69 };
70
71
72
73 #endif
```

7.10 Block.hpp

```
1 #ifndef GAME_BLOCK_HPP
2 #define GAME_BLOCK_HPP
3
4 #include <gameplay/PhysObject.hpp>
5 #include <gameplay/GameObjectTypes.hpp>
6 #include <framework/RenderSystem.hpp>
7 #include <box2d/b2_world.h>
8 #include <box2d/b2_body.h>
9 #include <box2d/b2_fixture.h>
```

```
10
12 class Block : public PhysObject {
13 public:
14
16     Block(Game& game, gm::GameObjectType type, float x, float y, float rot);
17
19     virtual void Render(const RenderSystem& r);
20
22     virtual void OnCollision(const b2Vec2& velocity, PhysObject& other, const b2Contact& contact);
23
25     virtual std::vector<sf::Sprite> GetSprites(const RenderSystem& r);
26
28     virtual bool CheckIntersection(sf::Sprite s, const RenderSystem& r);
29
31     virtual std::vector<b2Body*> GetPhysBodies();
32
34     virtual bool CheckIntersection(b2Body* other);
35
37     const gm::BlockMaterial GetBlockMaterial() const;
38
40     bool ElectricityCheck(Block& block);
41
42
43 protected:
44     gm::BlockData blockData_;
45     gm::BlockShapeData shapeData_;
46     gm::BlockMaterialData materialData_;
47     float lastHitSound_ = 0.0F;
48
49     virtual void OnDeath();
50 };
51
52
53
54 #endif
```

7.11 Camera.hpp

```
1 #ifndef CAMERA_HPP
2 #define CAMERA_HPP
3
4 #include <ui/UIConstants.hpp>
5 #include <gameplay/Physics.hpp>
6
7 /* A Camera has a position in world space, and a zoom.
8  * zoom < 1 means zooming in, zoom > 1 means zooming out.
9  * zoom = 1 is fullscreen.
10  *
11  * As per the definition in gameplay/Physics.hpp, a camera at fullscreen zoom
12  * will see an area that is 50 meters wide.
13  */
14
16
24 struct Camera {
25
27     float x = 0;
29     float y = 0;
31     float rot = 0;
33     float zoom = 1;
34
36     void SetFullscreen() {
37         x = 0;
38         y = 0.5F * ph::fullscreenPlayArea / ui::aspectRatio - ph::groundThickness;
39         rot = 0;
40         zoom = 1;
41     }
42 };
43
44
45
46 #endif
```

7.12 Cannon.hpp

```
1 #ifndef CATAPULT_HPP
2 #define CATAPULT_HPP
3 #define _USE_MATH_DEFINES
4 #include <gameplay/Physics.hpp>
5 #include <gameplay/GameObject.hpp>
```

```
6 #include <gameplay/GameObjectTypes.hpp>
7 #include <framework/RenderSystem.hpp>
8 #include <framework/Resources.hpp>
9 #include <cmath>
10 #include <SFML/System/Vector2.hpp>
11
12
15 class Cannon : public GameObject {
16 public:
17
19     Cannon(Game& game, gm::GameObjectType type, float x, float y, float rot);
20
22     virtual ~Cannon();
23
25     virtual void Update();
26
28     virtual void Render(const RenderSystem& r);
29
31     virtual bool OnMouseMove(float xw, float yh);
32
34     virtual bool OnMouseDown(const sf::Mouse::Button& e, float x, float y);
35
37     virtual bool OnMouseUp(const sf::Mouse::Button& e, float x, float y);
38
39
40 private:
41     bool isActive_;
42     const float sizeh_ = 1;
43
44     float x_base_;
45     float y_base_;
46     float h_base_;
47     const float rot_base_ = 0;
48
49     float x_pipe_;
50     float y_pipe_;
51     float h_pipe_;
52     float rot_pipe_;
53
54     float x_loadBar_;
55     float y_loadBar_;
56     float h_loadBar_;
57     float w_loadBar_;
58     const sf::Color barColor_ = {255, 0, 0};
59
60     sf::Vector2f relativeCoords_;
61     float relativeDistance_ = 0.5F;
62 };
63
64
65
66 #endif
```

7.13 Editor.hpp

```
1 #ifndef EDITOR_HPP
2 #define EDITOR_HPP
3
4 #include <gameplay/Game.hpp>
5 #include <screens/GameScreen.hpp>
6
8 class Editor: public Game{
9 public:
11     Editor(GameScreen &s, Level level);
12
14     void SetSelectedElement(gm::GameObjectType t);
15
17     void AddProjectile(gm::GameObjectType t);
18
20     void RemoveProjectile(std::size_t index);
21
23     virtual void Resume();
24
26     virtual void Restart();
27
29     bool InPlayMode() const;
30
32     virtual bool OnMouseMove(float xw, float yh);
34     virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh);
36     virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh);
38     virtual bool OnKeyDown(const sf::Event::KeyEvent&);
39
41     void Play();
```

```
42
44      Level& GetLevel();
45
47      void SaveLevel();
48
50      virtual bool IsEditor() const { return true; }
51
52  private:
53      gm::GameObjectType selectedElement_;
54      int dragObjectID_ = -1;
55      ph::tfloat dragX_;
56      ph::tfloat dragY_;
57
58      bool playMode_ = false;
59
60  };
61
62  #endif
```

7.14 Effect.hpp

```
1  #ifndef EFFECT_HPP
2  #define EFFECT_HPP
3  #include <framework/Resources.hpp>
4  #include <gameplay/GameObjectTypes.hpp>
5  #include <gameplay/GameObject.hpp>
6  #include <gameplay/Physics.hpp>
7  #include <iostream>
8
10 class Effect : public GameObject {
11 public:
13      Effect(Game& game, AnimationID anim, float x, float y, float rot, float size = 1.0F, float fps =
        24.0F, float duration = 1.0F, bool loop = false) :
14      GameObject(game, gm::GameObjectType::anim_effect, x, y, rot), fps_(fps), size_(size),
        duration_(duration), loop_(loop), animationID_(anim)
15      {starting_time = game_.GetTime();}
16
18      int GetFrame(){
19          float time = game_.GetTime() - starting_time;
20          return (int)(time * fps_);
21      }
23      bool CheckDuration(){
24          if(game_.GetTime() - starting_time > duration_){
25              return true;
26          }
27          return false;
28      }
30      virtual void Render(const RenderSystem& r) {
31
32          int frame = CheckDuration() ? (int)(duration_ * fps_) : GetFrame();
33          if(loop_) frame = GetFrame();
34          r.RenderAnimation(animationID_, frame, x_, y_, size_, rot_, game_.GetCamera());
35
36      }
38      virtual void Update() {
39          if(CheckDuration()) game_.DestroyObject(gameID_);
40      }
41
42 private:
43
44      float fps_;
45      float size_;
46      float starting_time;
47      float duration_;
48      bool loop_;
49      AnimationID animationID_;
50 };
51
52 #endif
```

7.15 Fuksi.hpp

```
1  #ifndef FUKSI_HPP
2  #define FUKSI_HPP
3
4  #include <gameplay/GameObjectTypes.hpp>
5  #include <gameplay/ParticleEffect.hpp>
6  #include <gameplay/PhysObject.hpp>
7  #include <gameplay/Physics.hpp>
```

```
8 #include <gameplay/Person.hpp>
9 #include <box2d/b2_body.h>
10 #include <memory>
11
13 class Fuksi : public Person {
14 public:
16     Fuksi(Game& game, float x, float y, float rot, gm::PersonData data) : Fuksi(game, x, y, rot) { data_
       = data; }
17     Fuksi(Game& game, float x, float y, float rot) : Person(game, gm::GameObjectType::fuksi, x, y, rot,
       true, -6)
18     { data_ = gm::RandomFuksi(); hp_ = ph::fuksiHP; }
19
20 protected:
22     virtual void OnDeath() {
23         //Spawn points effect
24         int points = gm::GetObjectScore(objectType_);
25         int id = game_.AddObject(std::make_unique<TextParticle>(game_, x_, y_, 0.0F));
26         TextParticle& textP = (TextParticle&)game_.GetObject(id);
27         textP.SetSize(1.0F);
28         textP.SetColor(sf::Color(0, 0, 255, 255));
29         std::stringstream ss;
30         ss « points;
31         textP.SetText(ss.str());
32
33         game_.AddPoints(gm::GetObjectScore(gm::GameObjectType::fuksi));
34         game_.CheckLevelEnd();
35     }
36
37 private:
38
39 };
40
41
42 #endif
```

7.16 Game.hpp

```
1 #ifndef GAME_HPP
2 #define GAME_HPP
3
4 #include <memory>
5 #include <UpdateListener.hpp>
6
7 #include <gameplay/Camera.hpp>
8 #include <gameplay/Level.hpp>
9 #include <gameplay/GameObject.hpp>
10 #include <gameplay/GameObjectTypes.hpp>
11 #include <framework/AudioSystem.hpp>
12
13 #include <box2d/b2_world.h>
14 #include <box2d/b2_world_callbacks.h>
15 #include <box2d/b2_body.h>
16 #include <box2d/b2_contact.h>
17 #include <SFML/System/Vector2.hpp>
18
19 #include <iostream>
20
23 struct IDCounter {
24     int backgrounds = 0;
25     int blocks = 1 * gm::objectGroupSize;
26     int teekkaris = 2 * gm::objectGroupSize;
27     int effects = 3 * gm::objectGroupSize;
28 };
29
30 //Forward declaration
31 class GameScreen;
32
33
35
55 class Game : public UpdateListener, public b2ContactListener {
56 public:
57
59     Game(GameScreen&);
60
62     Game(GameScreen &s, Level level);
63
64     virtual ~Game();
65
67     virtual void Render(const RenderSystem& r);
68
70     virtual void Update();
71
73     void Pause();
```

```
74
76      virtual void Resume();
78      void Restart();
79
81      void LoadLevel(Level level);
83      int GetMaxScore();
84
85
87
95      int AddObject(std::unique_ptr<GameObject>);
96
98      int CreateObject(gm::GameObjectData data);
99
101      int CreateObject(gm::GameObjectType type, float x = 0, float y = 0, float rot = 0);
102
104      int CreateTeekkari(gm::PersonData data, float x = 0, float y = 0, float rot = 0);
105
107      void DestroyObject(int id);
108
110      void ClearObjects();
111
113      GameObject& GetObject(int id);
114
116      std::vector<GameObject*> GetObjects();
117
118
120      unsigned int GetTicks() const;
121
123      float GetTime() const;
124
126      float GetTimeForUI() const;
127
129      bool IsPaused() const;
130
132      bool CannonDisabled() const;
133
135      AudioSystem& GetAudioSystem() const;
136
138      b2World& GetB2World();
139
141      GameScreen& GetScreen();
142
144      virtual void BeginContact(b2Contact* contact);
145
146
148      const Camera& GetCamera() const;
149
151      void ResetCamera();
152
154      void SetCameraPos(float x, float y);
155
157      void SetCameraZoom(float zoom);
158
160      void SetCameraRot(float rot);
161
163      void CheckLevelEnd();
164
166      void AddPoints(int p);
167
169      void AddTeekkari(gm::GameObjectType teekkari);
170
172      void ProfessorPause();
173
175      void ProfessorResume();
176
178      void SelectProjectile(int index);
179
181      bool TakeProjectile(gm::PersonData& teekkari);
182
184      virtual bool OnMouseMove(float xw, float yh);
186      virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh);
188      virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh);
190      virtual bool OnMouseScroll(float delta, float xw, float yh);
191
192
194      bool NoFuksis();
195
197      bool NoTeekkaris();
198
200      bool NoActivity();
201
203      virtual bool IsEditor() const { return false; }
204
205      //std::map<int,std::unique_ptr<GameObject>>& GetObjects();
206
207  protected:
208      GameScreen& screen_;
```

```
209
210     IDCounter IDCounter_;
211     std::map<int,std::unique_ptr<GameObject» objects_;
212     Level level_;
213     Camera camera_;
214
216     std::vector<gm::PersonData> teekkarisLeft_;
217     int chosenTeekkari_ = 0;
218
219     //Mark this level for ending check. It is important that the level isn't ended in the middle of an
        Update, since
220     //that could cause us to reference destructed objects
221     bool checkForFinish_ = false;
222
224     bool professorPause_ = false;
225
226     bool isPaused_ = false;
227     int points_;
228     unsigned int time_ = 0; //Game ticks since starting => number of update calls
229
230     int levelMaxScore_ = 0;
231
232     b2World world_;
233
235     void UpdateProjectileList();
236
237
238     //Variables for panning the camera
239     bool movingCamera_ = false;
240     ph::tfloat cameraGrabX_;
241     ph::tfloat cameraGrabY_;
242
244     void CheckCameraBounds();
245
246 };
247
248
249 #endif
```

7.17 GameObject.hpp

```
1 #ifndef GAME_OBJECT_HPP
2 #define GAME_OBJECT_HPP
3
4 #include <UpdateListener.hpp>
5 #include <gameplay/Game.hpp>
6 #include <gameplay/Physics.hpp>
7 #include <gameplay/GameObjectTypes.hpp>
8 #include <box2d/b2_body.h>
9 #include <vector>
11 class GameObject : public UpdateListener {
12 public:
13
15     GameObject();
16
18     GameObject(Game& game, gm::GameObjectType objectType, float x, float y, float rot) :
19     game_(game), objectType_(objectType), x_(x), y_(y), rot_(rot) {}
20
22     virtual ~GameObject() = default;
23
25     virtual void Record() {
26         x_.Record();
27         y_.Record();
28         rot_.Record();
29     }
30
31     //Important that these are virtual.
32     //For example, PhysObject derives these to set its underlying rigidbody position
33
35     virtual void SetX(float x) { x_ = x; }
36
38     virtual void SetY(float y) { y_ = y; }
39
41     virtual void SetRotation(float rot) { rot_ = rot; }
42
44     virtual void SetPosition(float x, float y) { this->SetX(x); this->SetY(y); }
45
47     virtual ph::tfloat GetX() const { return x_; }
48
50     virtual ph::tfloat GetY() const { return y_; }
51
53     virtual ph::tfloat GetRot() const { return rot_; }
54
```

```
56      gm::GameObjectType GetObjectType() const { return objectType_; }
57
59      int GetGameID() const { return gameID_; }
60
62      virtual void Render(const RenderSystem&) = 0;
63
65      virtual std::vector<sf::Sprite> GetSprites(const RenderSystem& r) { return std::vector<sf::Sprite>();
        }
66
68      virtual bool CheckIntersection(sf::Sprite s, const RenderSystem& r) { return false; }
69
71      virtual bool ContainsCoordinates(sf::Vector2f mouseCoords, const RenderSystem& r) { return false; }
73      virtual std::vector<b2Body*> GetPhysBodies() { return std::vector<b2Body*>(); }
75      virtual bool CheckIntersection(b2Body* other) { return false; }
76
77  protected:
78
79      //Allow Game to modify gameID_ when taking ownership of an object
80      friend class Game;
81
82      Game& game_;
83      ph::tfloat x_;
84      ph::tfloat y_;
85      ph::tfloat rot_;
86      gm::GameObjectType objectType_; //Object type
87      int gameID_ = -1;               //Object id
88  };
89
90
91  #endif
```

7.18 GameObjectTypes.hpp

```
1  #ifndef GAME_OBJECT_TYPES
2  #define GAME_OBJECT_TYPES
3
4  #include <map>
5  #include <memory>
6  #include <utility>
7  #include <vector>
8  #include <unordered_map>
9  #include <box2d/b2_shape.h>
10 #include <box2d/b2_polygon_shape.h>
11 #include <box2d/b2_circle_shape.h>
12 #include <framework/Resources.hpp>
13 #include <string>
14
15
16 class Game;
17 class GameObject;
18
20 namespace gm {
21
22 //List of all game object types
23 //Defined in the cpp file
24
26 enum GameObjectGroup {
27      background,
28      block,
29      teekkari,
30      effect,
31      ground
32 };
33
35 enum GameObjectType {
36
37
38      //Background objects
39      terrain1x1, //Unmovable block of terrain
40      background_tree1,
41      background_tree2,
42      background_lamp_pole,
43      background_bench,
44      background_person1,
45      background_person2,
46      background_person3,
47
48
49      //Blocks
50      block_wood1x1,
51      block_metal1x1,
52      block_glass1x1,
53      block_plastic1x1,
```

```
54      block_rubber1x1,
55      block_concrete1x1,
56
57      block_wood2x1,
58      block_metal2x1,
59      block_glass2x1,
60      block_plastic2x1,
61      block_rubber2x1,
62      block_concrete2x1,
63
64      block_wood2x2,
65      block_metal2x2,
66      block_glass2x2,
67      block_plastic2x2,
68      block_rubber2x2,
69      block_concrete2x2,
70
71      ball_wood,
72      ball_metal,
73      ball_glass,
74      ball_plastic,
75      ball_rubber,
76      ball_concrete,
77
78      /*
79      block_woodTri,
80      block_metalTri,
81      block_glassTri,
82      block_plasticTri,
83      block_rubberTri,
84      block_concreteTri,*/
85
86      plank_wood,
87      plank_metal,
88      plank_glass,
89      plank_plastic,
90      plank_rubber,
91      plank_concrete,
92
93      thickplank_wood,
94      thickplank_metal,
95      thickplank_glass,
96      thickplank_plastic,
97      thickplank_rubber,
98      thickplank_concrete,
99
100
101      //Props
102      prop_beer,
103      prop_beer_can,
104      prop_chair,
105      prop_table,
106
107      prop_sofa2x1,
108      prop_tnt,
109
110      pickup_ik,
111      pickup_sik,
112      pickup_tefy,
113      pickup_tuta,
114      pickup_tik,
115      pickup_inkubio,
116      pickup_kik,
117      pickup_professor,
118
119
120      //Cannon
121      cannon,
122
123
124      //Teekkaris
125      teekkari_ik,
126      teekkari_sik,
127      teekkari_tefy,
128      teekkari_tuta,
129      teekkari_tik,
130      teekkari_inkubio,
131      teekkari_kik,
132      teekkari_professor,
133
134
135      //Fuksi (all fuksis are functionally identical, only different in appearance)
136      fuksi,
137
138      //Physics particle
139      phys_particle,
140      professor_particle,
```

```
141
142     //Animation
143     anim_effect,
144
145     //Teekkari abilities (those that have spawnable components)
146     ability_cow,
147     ability_wrench,
148     ability_integral,
149
150
151
152     ground_obj
153
154 };
155
157 struct GameObjectData {
158
159     float x;
160     float y;
161     float rot;
162     GameObjectType type;
163
164 };
165
166 const int objectGroupSize = 100000000;
167
169 int GetObjectGroup(GameObjectType);
170
172 int GetObjectScore(GameObjectType type);
173
175 std::unique_ptr<GameObject> IDToObject(Game& game, GameObjectType type, float x, float y, float rot);
176
177
178
179
181 struct PersonFace {
182     SpriteID face = SpriteID::teekkari_head1;
183     SoundID grunt = SoundID::grunt1;
184     SoundID die = SoundID::teekkari_death1;
185     bool bType = false; //use tanned hands because the face is also tanned
186
187     //just looks wrong if the face doesn't match the hands
188 };
189
191 struct PersonBody {
192     SpriteID torso = SpriteID::torso_blue;
193     SpriteID arm = SpriteID::arm_blue;
194     SpriteID leg = SpriteID::leg_blue;
195     SpriteID armb = SpriteID::armb_blue;
196     std::string guildName = "Teemu Teekkari";
197 };
199
202 struct PersonData {
203     PersonFace face;
204     PersonBody body;
205     GameObjectType objType = GameObjectType::teekkari_ik;
206 };
207
208
210 extern const std::vector<PersonFace> teekkariHeads;
211
213 extern const std::vector<PersonFace> teekkariHeads_s;
214
216 extern const std::vector<PersonFace> fuksiHeads;
217
219 extern const std::unordered_map<GameObjectType, PersonBody> teekkariBodies;
220
222 extern const std::vector<gm::PersonBody> fuksiBodies;
223
225 extern const std::vector<gm::GameObjectType> teekkaris;
226
228 extern const std::unordered_map<gm::GameObjectType, gm::GameObjectType> pickupLookup;
229
231 PersonData RandomTeekkari(GameObjectType type);
232
234 PersonData RandomFuksi();
235
236
237
239 enum BlockMaterial { wood, metal, glass, plastic, rubber, concrete };
241 enum BlockShape { block_1x1, block_2x1, block_2x2, block_ball, /*block_tri,*/ block_plank,
       block_thickplank, block_bottle, block_can };
242
243
244 // Allocate and create shared base shapes for BlockShapeData to use
245
246 std::shared_ptr<b2Shape> CreateShape1x1();
```

```
247 std::shared_ptr<b2Shape> CreateShape2x1();
248 std::shared_ptr<b2Shape> CreateShape2x2();
249 std::shared_ptr<b2Shape> CreateShapeBall();
250 //std::shared_ptr<b2Shape> CreateShapeTri();
251 std::shared_ptr<b2Shape> CreateShapePlank();
252 std::shared_ptr<b2Shape> CreateShapeThickPlank();
253 std::shared_ptr<b2Shape> CreateShapeBottle();
254 std::shared_ptr<b2Shape> CreateShapeCan();
255
257 struct BlockMaterialData {
258     BlockMaterial material;
259     float density;
260     float friction;
261     float restitution;
262     float hpMassRatio;
263     float pointsPerMass;
264     SoundID hitSound;
265     SoundID breakSound;
266     SpriteID particle;
267 };
268
270 struct BlockShapeData {
271     BlockShape shape;
272     float volume;
273     float height;
274     std::shared_ptr<b2Shape> b2shape;
275     SpriteID halfHPSprite;
276     SpriteID lowHPSprite;
277 };
278
280 struct BlockData {
281     std::string blockName;
282     SpriteID sprite;
283     BlockMaterial material;
284     BlockShape shape;
285 };
286
288 extern const std::map<GameObjectType, BlockData> blockTypes;
289
291 extern const std::unordered_map<BlockMaterial, BlockMaterialData> materialProperties;
292
294 extern const std::unordered_map<BlockShape, BlockShapeData> shapeProperties;
295
296
297 }
298
299 #endif
```

7.19 Ground.hpp

```
1 #ifndef GROUND_HPP
2 #define GROUND_HPP
3
4 #include <gameplay/PhysObject.hpp>
5 #include <gameplay/GameObjectTypes.hpp>
6 #include <gameplay/Physics.hpp>
7 #include <box2d/b2_polygon_shape.h>
8 #include <box2d/b2_body.h>
9 #include <box2d/b2_fixture.h>
10 #include <limits>
11
13 class Ground : public PhysObject {
14 public:
16     Ground(Game& game) : PhysObject(game, gm::GameObjectType::ground_obj, 0, -0.5F * ph::groundThickness,
       0) {
17         b2BodyDef groundDef;
18         b2PolygonShape groundShape;
19
20         groundDef.type = b2BodyType::b2_staticBody;
21         groundDef.position = {0, -0.5F * ph::groundThickness};
22         groundShape.SetAsBox(ph::fullscreenPlayArea * 0.5F, 0.5F * ph::groundThickness);
23
24         //Box2D clones the data so it doesn't matter that groundDef and groundShape go out of scope
25         mainBody_ = game.GetB2World().CreateBody(&groundDef);
26
27         b2FixtureDef fixture;
28         b2FixtureUserData userData;
29         userData.data = this;
30         fixture.density = 0;
31         fixture.shape = &groundShape;
32         fixture.userData = userData;
33
34         mainBody_->CreateFixture(&fixture);
```

```
35          hp_ = std::numeric_limits<float>::infinity();
36      }
38      virtual float GetMass() const { return ph::groundMass;}
40      virtual void Render(const RenderSystem& r) {
41          r.RenderRect(ph::groundColor, 0, -0.5F * ph::groundThickness, ph::fullscreenPlayArea,
        ph::groundThickness, 0, game_.GetCamera());
42      }
44      virtual void Update() {
45      }
46
47
48  };
49
50
51  #endif
```

7.20 Level.hpp

```
1  #ifndef LEVEL_HPP
2  #define LEVEL_HPP
3
4  #include <vector>
5  #include <string>
6  #include <utility>
7  #include <gameplay/GameObjectTypes.hpp>
8  #include <framework/Resources.hpp>
9
11 enum LevelMode { normal, time_trial, endless };
12 inline const std::vector<std::string> levelModeNames = {"normal", "time trial", "endless"};
13
15 struct Level {
17      std::string levelName = "new level";
19      std::string levelPath = "";
21      int timeLimit = 0;
23      int perfectScore = 0;
25      LevelMode levelMode = LevelMode::normal;
27      std::vector<gm::GameObjectData> objectData;
29      std::vector<std::pair<std::string, int» highscores;
31      SpriteID backgroundImage = SpriteID::background_field;
32
34      std::vector<gm::GameObjectType> startingTeekkaris;
35
37      int CalculateMaxScore() {
38
39          int sum = 0;
40          for(const auto& obj : objectData)
41              sum += gm::GetObjectScore(obj.type);
42          if(startingTeekkaris.empty()) return sum;
43          else return sum + (startingTeekkaris.size()-1) *
        gm::GetObjectScore(gm::GameObjectType::teekkari_ik);
44
45      }
46
47
48  };
49
50  #endif
```

7.21 ParticleEffect.hpp

```
1  #ifndef PARTICLE_EFFECT_HPP
2  #define PARTICLE_EFFECT_HPP
3
4  #include <gameplay/GameObjectTypes.hpp>
5  #include <gameplay/PhysObject.hpp>
6  #include <box2d/b2_circle_shape.h>
7  #include <box2d/b2_fixture.h>
8  #include <box2d/b2_body.h>
9  #include <limits>
10 #include <string>
11 #include <cmath>
12
14 class PhysParticle : public PhysObject {
15 public:
17      PhysParticle(Game& game, float x, float y, float rot) : PhysObject(game,
        gm::GameObjectType::phys_particle, x, y, rot) {
18          hp_ = std::numeric_limits<float>::infinity();
19          creationTime_ = game.GetTime();
20
```

```
21          //Create the main body
22          b2BodyDef definition;
23          definition.type = b2BodyType::b2_dynamicBody;
24          definition.fixedRotation = false;
25          definition.position = {x, y};
26          definition.angle = ph::rotToAng(rot);
27
28          mainBody_ = game.GetB2World().CreateBody(&definition);
29
30          b2CircleShape shape;
31          shape.m_radius = 0.1F;
32          b2FixtureDef fixture;
33          b2FixtureUserData userData;
34          fixture.density = 100.0F;
35          fixture.isSensor = true;
36          userData.data = this;
37          fixture.shape = &shape;
38          fixture.userData = userData;
39          mainBody_->CreateFixture(&fixture);
40
41          Record();
42      }
44      virtual void Render(const RenderSystem& r) {
45          float opacity = 1.0F;
46          float t = creationTime_ + lifeTime_ - game_.GetTime();
47          if(t < ph::particleFadeTime) {
48              opacity = t / ph::particleFadeTime;
49              opacity *= opacity;
50          }
51          r.RenderSprite(sprite_, x_, y_, size_, rot_, game_.GetCamera(), sf::Color(255, 255, 255,
    (int)std::roundf(opacity * 255)));
52      }
54      virtual void Update() {
55          if(game_.GetTime() - creationTime_ > lifeTime_) hp_ = 0;
56          PhysObject::Update();
57      }
59      void SetSize(float sz) { size_ = sz; }
61      void SetLifeTime(float l) { lifeTime_ = l; }
63      void SetSprite(SpriteID sp) { sprite_ = sp; }
65      virtual std::vector<b2Body*> GetPhysBodies() { return std::vector<b2Body*>(); }
66
68      b2Body* GetBody() { return mainBody_; }
69
70  protected:
71
72      // Allows professor to create special particles that move in stopped time
73      float creationTime_;
74      float size_ = 0.1F;
75      float lifeTime_ = 1.0F;
76      SpriteID sprite_ = SpriteID::particles_dust;
77  };
78
80  class TextParticle : public PhysObject {
81  public:
83      TextParticle(Game& game, float x, float y, float rot) : PhysObject(game,
    gm::GameObjectType::phys_particle, x, y, rot) {
84          hp_ = std::numeric_limits<float>::infinity();
85          creationTime_ = game.GetTime();
86
87          //Create the main body
88          b2BodyDef definition;
89          definition.type = b2BodyType::b2_dynamicBody;
90          definition.fixedRotation = false;
91          definition.position = {x, y};
92          definition.angle = ph::rotToAng(rot);
93
94          mainBody_ = game.GetB2World().CreateBody(&definition);
95
96          b2CircleShape shape;
97          shape.m_radius = 0.1F;
98          b2FixtureDef fixture;
99          b2FixtureUserData userData;
100         fixture.density = 100.0F;
101         fixture.isSensor = true;
102         userData.data = this;
103         fixture.shape = &shape;
104         fixture.userData = userData;
105         mainBody_->CreateFixture(&fixture);
106         mainBody_->SetGravityScale(-0.5F);
107
108         Record();
109     }
111     virtual void Render(const RenderSystem& r) {
112         float opacity = 1.0F;
113         float t = creationTime_ + lifeTime_ - game_.GetTime();
114         if(t < ph::particleFadeTime) {
115             opacity = t / ph::particleFadeTime;
```

```
116             opacity *= opacity;
117         }
118         r.RenderText(text_, x_, y_, size_, rot_, game_.GetCamera(), sf::Color(color_.r, color_.g,
    color_.b, (int)std::roundf(opacity * 255)), FontID::consolas);
119     }
121     virtual void Update() {
122         if(game_.GetTime() - creationTime_ > lifeTime_) hp_ = 0;
123         PhysObject::Update();
124     }
126     void SetSize(float sz) { size_ = sz; }
128     void SetLifeTime(float l) { lifeTime_ = l; }
130     void SetText(std::string text) { text_ = text; }
132     void SetColor(sf::Color color) { color_ = color; }
134     virtual std::vector<b2Body*> GetPhysBodies() { return std::vector<b2Body*>(); }
135
136 protected:
137     float creationTime_;
138     float size_ = 0.1F;
139     float lifeTime_ = 1.0F;
140     std::string text_ = "";
141     sf::Color color_ = {255, 106, 0, 255};
142 };
143
144
146 class ProfessorParticle : public GameObject {
147 public:
149     ProfessorParticle(Game& game, float x, float y, float rot) : GameObject(game,
    gm::GameObjectType::professor_particle, x, y, rot) {
150         creationTime_ = game.GetTime();
151         Record();
152     }
154     virtual void Render(const RenderSystem& r) {
155         float opacity = 1.0F;
156         float t = creationTime_ + lifeTime_ - GetRealTime();
157         if(t < ph::particleFadeTime) {
158             opacity = t / ph::particleFadeTime;
159             opacity *= opacity;
160         }
161         r.RenderSprite(sprite_, x_, y_, size_, rot_, game_.GetCamera(), sf::Color(255, 255, 255,
    (int)std::roundf(opacity * 255)));
162     }
164     virtual void Update() {
165         upd_++;
166         if(GetRealTime() - creationTime_ > lifeTime_) game_.DestroyObject(gameID_);
167         else {
168             Record();
169             x_ = x_ + ph::timestep * vx;
170             y_ = y_ + ph::timestep * vy;
171         }
172     }
174     void SetSize(float sz) { size_ = sz; }
176     void SetLifeTime(float l) { lifeTime_ = l; }
178     void SetSprite(SpriteID sp) { sprite_ = sp; }
180     virtual std::vector<b2Body*> GetPhysBodies() { return std::vector<b2Body*>(); }
182     void SetVelocity(float x, float y) { vx = x; vy = y; }
183
184
185 protected:
186
187     float GetRealTime() {
188         return creationTime_ + upd_ * ph::timestep;
189     }
190
191     int upd_ = 0;
192     float creationTime_;
193     float size_ = 0.1F;
194     float lifeTime_ = 1.0F;
195     SpriteID sprite_ = SpriteID::particles_dust;
196     float vx = 0;
197     float vy = 0;
198 };
199
200
201 #endif
```

7.22 Person.hpp

```
1 #ifndef PERSON_HPP
2 #define PERSON_HPP
3
4 #include <gameplay/GameObjectTypes.hpp>
5 #include <gameplay/PhysObject.hpp>
6 #include <gameplay/Physics.hpp>
```

```
7 #include <box2d/b2_body.h>
8
10 class Person : public PhysObject {
11 public:
12
14     Person(Game& game, gm::GameObjectType type, float x, float y, float rot, bool mirrored = false, int
       collisionGroup = -5);
15
17     virtual ~Person();
18
20     virtual void Render(const RenderSystem& r);
21
23     virtual float GetMass() const;
25     virtual void Record();
26
28     virtual void Update();
29
31     virtual void SetX(float x);
32
34     virtual void SetY(float y);
35
37     virtual void SetPosition(float x, float y);
38
40     virtual void SetRotation(float rot);
41
42
43     //SetPosition is same as GameObject::SetPosition
44
46     virtual void OnCollision(const b2Vec2& velocity, PhysObject& other, const b2Contact& contact);
48     virtual void Impulse(const b2Vec2& f);
49
50     virtual bool ContainsCoordinates(sf::Vector2f mouseCoords, const RenderSystem& r);
51     virtual std::vector<sf::Sprite> GetSprites(const RenderSystem& r);
52     virtual bool CheckIntersection(sf::Sprite s, const RenderSystem& r);
53
54     virtual std::vector<b2Body*> GetPhysBodies();
55     virtual bool CheckIntersection(b2Body* other);
56
57 protected:
58
59     // Data defining the look and sound of this person
60     gm::PersonData data_;
61
62     float lastHitSound_ = 0.0F;
63
64     inline static const float restitution = 0.3F;
65
66     inline static const float totalHeight = ph::personHeight;
67     inline static const float legHeight = 0.23913F * Person::totalHeight;
68     inline static const float armHeight = 1.13207F * Person::legHeight;
69     inline static const float torsoHeight = 1.47169F * Person::legHeight;
70     inline static const float headHeight = 2.064150F * Person::legHeight;
71
72     inline static const float torsoWidth = 0.8333F * Person::torsoHeight;
73     inline static const float legWidth = 0.6415F * Person::legHeight;
74     inline static const float armWidth = 0.56666F * Person::armHeight;
75     inline static const float headWidth = 0.89166F * Person::headHeight;
76
77     inline static const float torsoVolume = Person::torsoWidth * Person::torsoHeight;
78     inline static const float legVolume = Person::legWidth * Person::legHeight;
79     inline static const float armVolume = Person::armWidth * Person::armHeight;
80     inline static const float headVolume = 0.25F * Person::headHeight * Person::headHeight * ph::pi;
81
82     inline static const float totalVolume = Person::torsoVolume + 2 * Person::legVolume + 2 *
       Person::armVolume + Person::headVolume;
83
84     b2Body* headBody_;
85     b2Body* armRBody_;
86     b2Body* armLBody_;
87     b2Body* legRBody_;
88     b2Body* legLBody_;
89
90     ph::tfloat headX_;
91     ph::tfloat headY_;
92     ph::tfloat headRot_;
93
94     ph::tfloat armRX_;
95     ph::tfloat armRY_;
96     ph::tfloat armRRot_;
97
98     ph::tfloat armLX_;
99     ph::tfloat armLY_;
100     ph::tfloat armLRot_;
101
102     ph::tfloat legRX_;
103     ph::tfloat legRY_;
104     ph::tfloat legRRot_;
```

```
105
106     ph::tfloat legLX_;
107     ph::tfloat legLY_;
108     ph::tfloat legLRot_;
109
110
111 };
112
113 #endif
```

7.23 Physics.hpp

```
1 #ifndef PHYSICS_HPP
2 #define PHYSICS_HPP
3
4
5 #include <SFML/Graphics/Color.hpp>
6 #include <SFML/System/Vector2.hpp>
7 #include <cmath>
8 #include <limits>
9
11
65 namespace ph {
66
67
69     const float fullscreenPlayArea = 50.0F;
70
72     const float lightningEnergy = 800000.0F;
73
75     const float electricityJumpGap = 1.0F;
76
78     const float groundThickness = 20.0F;
79
81     const float cameraUpperBound = 50.0F;
82
84     const float groundMass = 100.0F;
85
87     const sf::Color groundColor = {98, 122, 31};
88
90     const float gravity = 9.81F;
91
93     const int velocityIters = 8;
94
96     const int positionIters = 3;
97
99     const float timestep = 0.02F; //50 updates per second
100
102     const float explosionDecay = 0.69314718F; //ln(2), half the power at 1 meter, fourth at 2...
103
105     const float collisionTreshold = 12.0F;
106
108     const float damageTreshold = 30.0F;
109
111     const float soundCooldown = 0.1F;
112
114     const float damageScaling = 0.18F;
115
117     const float particleFadeTime = 0.5F;
118
120     const float cannonMaxForce = 5000;
121
123     const float cannonX = -20;
124
126     const float personHeight = 1.8F;
127
129     const float personMass = 200.0F;
130
132     const float teekkariHP = 8000;
133
135     const float fuksiHP = 200;
136
138     const int fuksiScore = 4000;
139
141     const int teekkariScore = 12000;
142
144     const float pi = 3.141592741F;
145
146     const float inf = std::numeric_limits<float>::infinity();
147
149     inline float angToRot(float ang) { return -180 * ang / pi; }
150
152     inline float rotToAng(float rot) { return -pi * rot / 180; }
153
```

```
155     inline sf::Vector2f rotateVector(float x, float y, float rot) {
156         float rad = rotToAng(rot);
157         return { x * cosf(rad) - y * sinf(rad), x * sinf(rad) + y * cosf(rad) };
158     }
159 
161 
169     struct tfloat {
170     public:
171         tfloat() : f0(0), f1(0) {}
172         tfloat(const float& f) : f0(f), f1(f) {}
173         operator float() const { return f1; }
174         tfloat& operator=(const float& f) { f1 = f; return *this; }
175         tfloat& operator*=(const float& f) { f1 *= f; return *this; }
176         tfloat& operator/=(const float& f) { f1 /= f; return *this; }
177         tfloat& operator+=(const float& f) { f1 += f; return *this; }
178         tfloat& operator-=(const float& f) { f1 -= f; return *this; }
179 
180         float Lerp(float t) const { return f0 + (f1 - f0) * t; }
181         void Record() { f0 = f1; }
182         float f0;
183         float f1;
184     };
185 
186 
187 
188 }
189 
190 
191 #endif
```

7.24 PhysObject.hpp

```
1  #ifndef PHYS_OBJECT_HPP
2  #define PHYS_OBJECT_HPP
3  
4  #include <memory>
5  #include <gameplay/Game.hpp>
6  #include <gameplay/GameObjectTypes.hpp>
7  #include <gameplay/GameObject.hpp>
8  
9  #include <box2d/b2_world.h>
10 #include <box2d/b2_body.h>
11 #include <box2d/b2_fixture.h>
12 #include <box2d/b2_shape.h>
13 #include <box2d/b2_contact.h>
14 
16 class PhysObject : public GameObject {
17 public:
19     PhysObject(Game& game, gm::GameObjectType objectID, float x, float y, float rot) :
20     GameObject(game, objectID, x, y, rot) { }
21 
23     virtual ~PhysObject();
24 
26     virtual void Update();
27 
29     virtual void OnCollision(const b2Vec2& relativeVelocity, PhysObject& other, const b2Contact&
       contact);
30 
32     virtual void SetX(float x);
33 
35     virtual void SetY(float y);
36 
38     virtual void SetRotation(float rot);
39 
41     virtual void SetPosition(float x, float y);
42 
44     virtual void Impulse(const b2Vec2& f);
45 
47     virtual void Impulse(const b2Vec2& f, const b2Vec2& p);
48 
50     virtual void Force(const b2Vec2& f);
51 
53     virtual void Force(const b2Vec2& f, const b2Vec2& p);
54 
56     virtual void Torque(float t);
57 
59     virtual void Angular(float a);
60 
62     virtual void Explosion(const b2Vec2& center, float magnitude);
63 
65     void ExplosionDamage(const b2Vec2& center, float damage);
66 
68     virtual void DealDamage(float damage);
```

```
69
71      virtual float GetHP() const;
73      virtual float GetMass() const;
74
75      virtual bool ContainsCoordinates(sf::Vector2f mouseCoords, const RenderSystem& r);
76      virtual std::vector<b2Body*> GetPhysBodies();
77      virtual bool CheckIntersection(b2Body* other);
78
79
80
81  protected:
82
84      b2Body* mainBody_;
85      float hp_ = 0;
86
87
89      virtual void OnDeath() { }
90
91      SpriteID hitSp_ = SpriteID::hit_stars;
92      b2Vec2 hitPoint_ = {0, 0};
93      bool spawnHit_ = false;
94
95  };
96
97
98  #endif
```

7.25 Pickup.hpp

```
1  #ifndef GAME_PICKUP_HPP
2  #define GAME_PICKUP_HPP
3
4  #include <gameplay/Block.hpp>
5  #include <gameplay/PhysObject.hpp>
6  #include <gameplay/GameObjectTypes.hpp>
7  #include <framework/RenderSystem.hpp>
8  #include <framework/RandomGen.hpp>
9  #include <gameplay/ParticleEffect.hpp>
10 #include <box2d/b2_circle_shape.h>
11 #include <box2d/b2_fixture.h>
12 #include <box2d/b2_world.h>
13 #include <box2d/b2_body.h>
14 #include <box2d/b2_api.h>
15 #include <unordered_map>
16
18 class Pickup : public Block {
19 public:
21      Pickup(Game& game, gm::GameObjectType type, float x, float y, float rot) : Block(game, type, x, y,
        rot) {}
22
23 protected:
25      virtual void OnDeath() {
26
27          game_.GetAudioSystem().PlaySound(materialData_.breakSound);
28          game_.AddPoints(gm::GetObjectScore(objectType_));
29
30          //Spawn some particles
31
32          for(int i = 0; i < 10; i++) {
33              //Random point inside circle
34              float a = 2.0F * ph::pi * rng::RandomF();
35              float u = rng::RandomF() + rng::RandomF();
36              float r = (u > 1) ? 2 - u : u;
37              float x = shapeData_.height * r * cosf(a);
38              float y = shapeData_.height * r * sinf(a);
39              int id = game_.AddObject(std::make_unique<PhysParticle>(game_, x_ + x, y_ + y,
        ph::angToRot(a)));
40              PhysParticle& p = (PhysParticle&)game_.GetObject(id);
41
42              p.SetSize(0.25F);
43              p.SetSprite(materialData_.particle);
44              p.Angular(rng::RandomInt(0, 1) ? rng::RandomF() * 0.4F : -rng::RandomF() * 0.4F);
45              p.Explosion({x_, y_}, 30.0F);
46          }
47
48      //Spawn pickup effect
49      int id = game_.AddObject(std::make_unique<TextParticle>(game_, x_+0.5F, y_, 0.0F));
50      TextParticle& textP = (TextParticle&)game_.GetObject(id);
51      textP.SetSize(1.0F);
52      textP.SetColor(sf::Color(0, 0, 0, 255));
53      textP.SetText("+1");
54
55      int id2 = game_.AddObject(std::make_unique<PhysParticle>(game_, x_-0.5F, y_, 0.0F));
```

```
56      PhysParticle& physP = (PhysParticle&)game_.GetObject(id2);
57      physP.SetSize(1.0F);
58      physP.SetSprite(SpriteID::teekkari_head1);
59      physP.GetBody()->SetGravityScale(-0.5F);
60
61
62
63
64      game_.GetAudioSystem().PlaySound(SoundID::teekkari_recruit);
65      game_.AddTeekkari(gm::pickupLookup.at(objectType_));
66
67
68      }
69  };
70
71
72
73  #endif
```

7.26 Teekkari.hpp

```
1  #ifndef TEEKKARI_HPP
2  #define TEEKKARI_HPP
3
4  #include <framework/RenderSystem.hpp>
5  #include <screens/GameScreen.hpp>
6  #include <ui/UIConstants.hpp>
7  #include <gameplay/GameObjectTypes.hpp>
8  #include <gameplay/ParticleEffect.hpp>
9  #include <gameplay/PhysObject.hpp>
10 #include <framework/RandomGen.hpp>
11 #include <SFML/System/Vector2.hpp>
12 #include <gameplay/Effect.hpp>
13 #include <gameplay/Physics.hpp>
14 #include <gameplay/Person.hpp>
15 #include <box2d/b2_body.h>
16 #include <iostream>
17 #include <memory>
18
19 #include <gameplay/Block.hpp>
20 #include <limits>
21 #include <cmath>
22 #include <set>
23
25 class Teekkari : public Person {
26 public:
27      Teekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, data.objType,
       x, y, rot)
28      { data_ = data; }
29      Teekkari(Game& game, gm::GameObjectType type, float x, float y, float rot) : Person(game, type, x, y,
       rot, false, -5)
30      { data_ = gm::RandomTeekkari(type); hp_ = ph::teekkariHP; creationTime_ = game_.GetTime(); }
31
32      virtual void Update() {
33          if(mainBody_->GetLinearVelocity().Length() < 0.1F) sleepCounter_++;
34          else sleepCounter_ = 0;
35          if(game_.GetTime() - creationTime_ > 8.0F || sleepCounter_ > 10) hp_ = 0;
36          Person::Update();
37      }
38
39      virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh) {
40          if(!game_.IsPaused() && button == sf::Mouse::Button::Right && !abilityUsed_) {
41              this->Ability(xw, yh);
42              abilityUsed_ = true;
43              return true;
44          }   else return false;
45      }
46
47 protected:
48      virtual void OnDeath() { game_.CheckLevelEnd(); }
49      virtual void Ability(float x, float y) = 0;
50
51      float creationTime_;
52      bool abilityUsed_ = false;
53      int sleepCounter_ = 0;
54
55 };
56
57
58 //Abilities
59
61 class AbilityCow : public PhysObject {
62 public:
```

```
63      AbilityCow(Game& game, float x, float y, float rot) : PhysObject(game,
        gm::GameObjectType::ability_cow, x, y, rot) {
64          hp_ = 300000;
65          creationTime_ = game.GetTime();
66
67          //Create the main body
68          b2BodyDef definition;
69          definition.type = b2BodyType::b2_dynamicBody;
70          definition.fixedRotation = false;
71          definition.position = {x, y};
72          definition.angle = ph::rotToAng(rot);
73
74          mainBody_ = game.GetB2World().CreateBody(&definition);
75
76          b2PolygonShape shape;
77          shape.SetAsBox(1.333333F, 1.0F);
78          b2FixtureDef fixture;
79          b2FixtureUserData userData;
80          userData.data = this;
81          fixture.density = 1000.0F;
82          fixture.friction = 0.0F;
83          fixture.restitution = 0.4F;
84          fixture.filter.groupIndex = -5;
85          fixture.shape = &shape;
86          fixture.userData = userData;
87          mainBody_->CreateFixture(&fixture);
88
89          Angular(rng::RandomInt(0, 1) ? (10000 + rng::RandomF() * 10000) : (-10000 - rng::RandomF() *
        10000));
90          Impulse({0, 14000.0F});
91
92          game_.GetAudioSystem().PlaySound(SoundID::cow_moo);
93
94          Record();
95      }
96
97      virtual void Render(const RenderSystem& r) {
98          r.RenderSprite(SpriteID::cow, x_, y_, 2.0F, rot_, game_.GetCamera());
99      }
100
101      virtual void Update() {
102          if(game_.GetTime() - creationTime_ > 5.0F) hp_ = 0;
103          PhysObject::Update();
104      }
105
106 protected:
107      float creationTime_;
108      virtual void OnDeath() {
109
110          //Moo
111          //Spawn smoke
112
113          game_.GetAudioSystem().PlaySound(SoundID::cow_death);
114          game_.GetAudioSystem().PlaySound(SoundID::poof);
115          game_.AddObject(std::make_unique<Effect>(game_, AnimationID::particles_poof,
116          x_, y_, 0.0F, 2.0F, 60.0F, 0.2666666F));
117
118          game_.CheckLevelEnd();
119      }
120 };
121
123 class AbilityWrench : public PhysObject {
124 public:
125      AbilityWrench(Game& game, float x, float y, float rot) : PhysObject(game,
        gm::GameObjectType::ability_wrench, x, y, rot) {
126          creationTime_ = game.GetTime();
127
128          //Create the main body
129          b2BodyDef definition;
130          definition.type = b2BodyType::b2_dynamicBody;
131          definition.fixedRotation = false;
132          definition.position = {x, y};
133          definition.angle = ph::rotToAng(rot);
134
135          mainBody_ = game.GetB2World().CreateBody(&definition);
136
137          gm::BlockMaterialData metalData = gm::materialProperties.at(gm::metal);
138
139          b2PolygonShape shape;
140          shape.SetAsBox(0.5F, 0.16F);
141          b2FixtureDef fixture;
142          b2FixtureUserData userData;
143          userData.data = this;
144          fixture.density = metalData.density * 0.04F;
145          fixture.friction = metalData.friction;
146          fixture.restitution = metalData.restitution;
147          fixture.filter.groupIndex = -5;
```

```
148         fixture.shape = &shape;
149         fixture.userData = userData;
150         mainBody_->CreateFixture(&fixture);
151 
152         hp_ = 1;
153 
154         Angular(40.0F);
155 
156         Record();
157 
158         game_.GetAudioSystem().PlaySound(SoundID::wrench_swish);
159 
160     }
161 
162     virtual void Render(const RenderSystem& r) {
163         r.RenderSprite(SpriteID::wrench, x_, y_, 0.32F, rot_, game_.GetCamera());
164     }
165 
166     virtual void Update() {
167         if(game_.GetTime() - creationTime_ > 5.0F) hp_ = 0;
168         PhysObject::Update();
169     }
170 
171 protected:
172     float creationTime_;
173     virtual void OnDeath() {
174 
175         game_.GetAudioSystem().PlaySound(SoundID::metal_hit);
176         game_.CheckLevelEnd();
177 
178     }
179 
180     virtual void OnCollision(const b2Vec2& velocity, PhysObject& other, const b2Contact& contact) {
181 
182         PhysObject::OnCollision(velocity, other, contact);
183         hp_ = 0;
184 
185         //Collision sound
186 
187         //Deal extra damage to wood
188         if(gm::GetObjectGroup(other.GetObjectType()) == gm::GameObjectGroup::block) {
189             if(gm::blockTypes.at(other.GetObjectType()).material == gm::BlockMaterial::wood) {
190                 other.DealDamage(1000);
191             }
192         }
193     }
194 
195 };
196 
197 
199 class AbilityIntegral : public GameObject {
200 public:
201     AbilityIntegral(Game& game, float x, float y, float rot) : GameObject(game,
        gm::GameObjectType::ability_integral, x, y, rot) {
202         creationTime_ = game.GetTime();
203         hitShape_.SetAsBox(width_ / 2, height_ / 2);
204         Record();
205     }
206 
207     virtual void Render(const RenderSystem& r) {
208         r.RenderSprite(SpriteID::integral_sign, x_, y_, height_, rot_, game_.GetCamera());
209     }
210 
211     virtual void Update() {
212 
213         updCount_++;
214 
215         if(GetRealTime() - creationTime_ > 3.2F) {
216             game_.DestroyObject(gameID_);
217         }
218         else {
219 
220             b2Transform hitTransform;
221             hitTransform.Set({x_, y_}, 0);
222 
223             auto objs = game_.GetObjects();
224             for(auto o : game_.GetObjects()) {
225                 if(o->GetObjectType() != gm::GameObjectType::ability_integral
226                 && o->GetObjectType() != gm::GameObjectType::teekkari_professor
227                 && o->GetObjectType() != gm::GameObjectType::professor_particle) {
228                     if(gm::GetObjectGroup(o->GetObjectType()) == gm::GameObjectGroup::block
229                     || gm::GetObjectGroup(o->GetObjectType()) == gm::GameObjectGroup::teekkari) {
230                         auto physBodies = o->GetPhysBodies();
231                         bool hit = false;
232                         for(auto p : physBodies) {
233                             if(b2TestOverlap(&hitShape_, 0, p->GetFixtureList()[0].GetShape(), 0,
        hitTransform, p->GetTransform())) {
```

```
234                                 hit = true;
235                                 break;
236                             }
237                         }
238                         if(hit) {
239                             PhysObject* phys = (PhysObject*)o;
240                             phys->DealDamage(std::numeric_limits<float>::infinity());
241                         }
242                     }
243                 }
244             }
245 
246             x_.Record();
247             x_ += ph::timestep * ph::fullscreenPlayArea / 3.2F;
248         }
249     }
250 
251 protected:
252 
253     float GetRealTime() {
254         return creationTime_ + updCount_ * ph::timestep;
255     }
256 
257     int updCount_ = 0;
258     float creationTime_;
259     b2PolygonShape hitShape_;
260     float height_ = 2.0F;
261     float width_ = 5.15625F;
262 
263 };
264 
265 
266 
267 //Teekkaris
269 class IKTeekkari : public Teekkari {
270 public:
271     IKTeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y, rot,
    data) { }
272     IKTeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
    gm::GameObjectType::teekkari_ik, x, y, rot) { }
273 
274     virtual void OnCollision(const b2Vec2& velocity, PhysObject& other, const b2Contact& contact) {
275 
276         float h = hp_;
277         Teekkari::OnCollision(velocity, other, contact);
278 
279         //Deal extra damage to all materials
280         if(gm::GetObjectGroup(other.GetObjectType()) == gm::GameObjectGroup::block) {
281             other.Impulse({-velocity.x * other.GetMass(), -velocity.y * other.GetMass()});
282             other.DealDamage(1200);
283 
284         }
285     }
286 
287 protected:
288     virtual void Ability(float x, float y) { }
289 };
291 class SIKTeekkari : public Teekkari {
292 public:
293     SIKTeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y,
    rot, data) {}
294     SIKTeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
    gm::GameObjectType::teekkari_ik, x, y, rot) {}
295 
296     virtual void Render(const RenderSystem& r) {
297         Teekkari::Render(r);
298 
299         if(abilityUsed_ && !used_) {
300             float t = game_.GetTime() - abilityStartTime_;
301             int frame = (int)(t * 20.0F);
302             r.RenderAnimation(AnimationID::thunder_sparks, frame, x_, y_, 2.0F, 0.0F,
    game_.GetCamera());
303         }
304 
305         if(lightning_) {
306             r.RenderSprite(SpriteID::lightning_strike, lightningPos_.x, lightningPos_.y, lightningH_,
    lightningRot_, game_.GetCamera());
307             if(game_.GetTime() > lightningStart_ + 0.05F) lightning_ = false;
308         }
309     }
310 
311     virtual void Update() {
312 
313         if(abilityUsed_ && !used_ && game_.GetTime() > abilityStartTime_ + 1.0F) {
314             ActiveAbility();
315             used_ = true;
316         }
```

```
317
318         int c = sleepCounter_;
319         Teekkari::Update();
320
321         //Prevent the Teekkari from despawning when he is charging his ability
322         if(abilityUsed_ && !used_) sleepCounter_ = c;
323     }
324
325 protected:
326     virtual void Ability(float x, float y) {
327         abilityStartTime_ = game_.GetTime();
328         game_.GetAudioSystem().PlaySound(SoundID::thunder_static);
329     }
330
331
332
333     float abilityStartTime_ = 0;
334     bool used_ = false;
335
336     b2Vec2 lightningPos_ = {0, 0};
337     float lightningH_ = 0;
338     float lightningRot_ = 0;
339     bool lightning_ = false;
340     float lightningStart_ = 0;
341
342     void ActiveAbility() {
343         std::set<int> metalBlocks;
344         int nearestBlock = -1;
345         float minDistance = ph::inf;
346         for(auto o : game_.GetObjects()) {
347             if(o->GetGameID() >= gm::objectGroupSize && o->GetGameID() < 2*gm::objectGroupSize) {
348                 Block& block = static_cast<Block&>(*o);
349                 if(block.GetBlockMaterial() == gm::BlockMaterial::metal) {
350                     metalBlocks.insert(o->GetGameID());
351                     float distance =
    std::sqrt((x_-block.GetX())*(x_-block.GetX())+(y_-block.GetY())*(y_-block.GetY()));
352                     if(distance < minDistance) {
353                         minDistance = distance;
354                         nearestBlock = o->GetGameID();
355                     }
356                 }
357             }
358         }
359         std::function<void(int,float)> destroyRecursively = [&](int startBlock, float remainingEnergy) {
360             Block& currentBlock = static_cast<Block&>(game_.GetObject(startBlock));
361             metalBlocks.erase(startBlock);
362             std::set<int>::iterator it = metalBlocks.begin();
363             while(it != metalBlocks.end()) {
364                 Block& nextBlock = static_cast<Block&>(game_.GetObject(*it));
365                 if(currentBlock.ElectricityCheck(nextBlock) && remainingEnergy > 200) {
366                     float exponentialDecay = 2.0F;
367                     destroyRecursively(*it,remainingEnergy/exponentialDecay);
368                     it = metalBlocks.begin();
369                 }
370                 else {
371                     ++it;
372                 }
373             }
374             currentBlock.DealDamage(remainingEnergy);
375             game_.AddObject(std::make_unique<Effect>(game_, AnimationID::lightning, currentBlock.GetX(),
    currentBlock.GetY(), 0.0F, 4.0F, 40.0F, 0.1F));
376         };
377         if(nearestBlock != -1 && minDistance < 15.0F) {
378             auto& block = game_.GetObject(nearestBlock);
379             b2Vec2 v = {block.GetX() - x_, block.GetY() - y_};
380             float d = v.Length();
381
382             lightning_ = true;
383             lightningStart_ = game_.GetTime();
384             lightningPos_ = {x_ + v.x * 0.5F, y_ + v.y * 0.5F};
385             lightningH_ = 0.364583333F * (v.Length());
386             lightningRot_ = (v.y > 0) ? acosf(v.x / d) : 2 * ph::pi - acosf(v.x / d);
387             lightningRot_ = ph::angToRot(lightningRot_);
388
389             destroyRecursively(nearestBlock,ph::lightningEnergy);
390             game_.GetAudioSystem().PlaySound(SoundID::thunder_strike);
391         }
392     }
393
394 };
396 class TEFYTeekkari : public Teekkari {
397 public:
398     TEFYTeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y,
    rot, data) {}
399     TEFYTeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
    gm::GameObjectType::teekkari_ik, x, y, rot) {}
400
```

```
401
402     virtual void Update() {
403         float t = 3.0F + abilityStartTime_ - game_.GetTime();
404         if(abilityUsed_ && t > 0) {
405             Torque(-8000.0F / (t*t));
406             if(gCounter == 0) {
407                 auto objs = game_.GetObjects();
408
409                 b2CircleShape circle;
410                 circle.m_radius = 15.0F;
411
412                 for(auto o : objs) {
413                     if(o->GetGameID() != gameID_) {
414                         auto physBodies = o->GetPhysBodies();
415                         bool hit = false;
416                         for(auto p : physBodies) {
417                             if(b2TestOverlap(&circle, 0, p->GetFixtureList()[0].GetShape(), 0,
    mainBody_->GetTransform(), p->GetTransform())) {
418                                 hit = true;
419                                 break;
420                             }
421                         }
422                         if(hit) {
423                             PhysObject* phys = (PhysObject*)o;
424                             phys->Explosion({x_, y_}, phys->GetMass() * -15.0F);
425                         }
426                     }
427                 }
428             }
429             gCounter++;
430             gCounter %= 5;
431         }
432
433         else if(abilityUsed_) {
434             game_.GetAudioSystem().PlaySound(SoundID::gravity_shiftdown);
435             hp_ = 0;
436         }
437         Teekkari::Update();
438     }
439
440     virtual void Render(const RenderSystem& r) {
441
442         float t = 3.0F + abilityStartTime_ - game_.GetTime();
443         if(abilityUsed_ && t > 0) {
444
445         float timeLeft = game_.GetTime() - abilityStartTime_;
446         int frame = (int)(timeLeft * 30.0F);
447
448
449         float fade = 200 * (t / 2.5F);
450         if(fade < 0) fade = 0;
451         sf::Color c = sf::Color(255, 255, 255, (int)roundf(fade));
452         r.RenderAnimation(AnimationID::gravity_spiral, frame, x_, y_, 2.5F, 0, game_.GetCamera(), c);
453         SpriteID arm = data_.face.bType ? data_.body.armb : data_.body.arm;
454         r.RenderSprite(arm, armLX_, armLY_, armHeight, armLRot_, game_.GetCamera(), c);
455         r.RenderSprite(data_.body.leg, legLX_, legLY_, legHeight, legLRot_, game_.GetCamera(), c);
456         r.RenderSprite(data_.body.torso, x_, y_, torsoHeight, rot_, game_.GetCamera(), c);
457         r.RenderSprite(data_.body.leg, legRX_, legRY_, legHeight, legRRot_, game_.GetCamera(), c);
458         r.RenderSprite(arm, armRX_, armRY_, armHeight, armRRot_, game_.GetCamera(), c);
459         r.RenderSprite(data_.face.face, headX_, headY_, headHeight, headRot_, game_.GetCamera(), c);
460         }
461         else Teekkari::Render(r);
462     }
463
464 protected:
465     virtual void Ability(float x, float y) {
466         game_.GetAudioSystem().PlaySound(SoundID::gravity_shiftup);
467
468         int id = game_.AddObject(std::make_unique<PhysParticle>(game_, x_, y_, 0.0F));
469         PhysParticle& p = (PhysParticle&)game_.GetObject(id);
470
471         p.SetSprite(SpriteID::gravity_symbols);
472         p.SetSize(1.5F);
473         p.GetBody()->SetGravityScale(0);
474         p.GetBody()->ApplyLinearImpulseToCenter({0, 10.0F}, true);
475
476         abilityStartTime_ = game_.GetTime();
477
478         mainBody_->SetGravityScale(0);
479         headBody_->SetGravityScale(0);
480         armRBody_->SetGravityScale(0);
481         armLBody_->SetGravityScale(0);
482         legRBody_->SetGravityScale(0);
483         legLBody_->SetGravityScale(0);
484
485         mainBody_->SetLinearDamping(3);
486         headBody_->SetLinearDamping(3);
```

```
487         armRBody_->SetLinearDamping(3);
488         armLBody_->SetLinearDamping(3);
489         legRBody_->SetLinearDamping(3);
490         legLBody_->SetLinearDamping(3);
491 
492         mainBody_->GetFixtureList()[0].SetSensor(true);
493         headBody_->GetFixtureList()[0].SetSensor(true);
494         armRBody_->GetFixtureList()[0].SetSensor(true);
495         armLBody_->GetFixtureList()[0].SetSensor(true);
496         legRBody_->GetFixtureList()[0].SetSensor(true);
497         legLBody_->GetFixtureList()[0].SetSensor(true);
498 
499 
500 
501         hp_ = std::numeric_limits<float>::infinity();
502     }
503 
504 
505 
506     int gCounter = 0;
507     float abilityStartTime_ = 0;
508 };
510 class TUTATeekkari : public Teekkari {
511 public:
512     TUTATeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y,
    rot, data) {}
513     TUTATeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
    gm::GameObjectType::teekkari_ik, x, y, rot) {}
514 
515     virtual void Update() {
516         if(abilityUsed_ && game_.GetTime() < abilityStartTime_ + 2.0F) {
517             Force({0, 2000.0F});
518             if(whooshCounter == 0) {
519                 game_.GetAudioSystem().PlaySound(SoundID::hand_whoosh);
520 
521                 auto objs = game_.GetObjects();
522 
523                 b2CircleShape circle;
524                 circle.m_radius = 4.0F;
525 
526                 for(auto o : objs) {
527                     if(o->GetGameID() != gameID_) {
528                         auto physBodies = o->GetPhysBodies();
529                         bool hit = false;
530                         for(auto p : physBodies) {
531                             if(b2TestOverlap(&circle, 0, p->GetFixtureList()[0].GetShape(), 0,
    mainBody_->GetTransform(), p->GetTransform())) {
532                                 hit = true;
533                                 break;
534                             }
535                         }
536                         if(hit) {
537                             PhysObject* phys = (PhysObject*)o;
538                             phys->ExplosionDamage({x_, y_}, 3000);
539                             phys->Explosion({x_, y_}, phys->GetMass() * 15.0F);
540                         }
541                     }
542                 }
543             }
544             whooshCounter++;
545             whooshCounter %= 5;
546 
547             armRBody_->SetAngularVelocity(100);
548             armLBody_->SetAngularVelocity(100);
549         }
550         Teekkari::Update();
551     }
552 
553     virtual void Render(const RenderSystem& r) {
554         Teekkari::Render(r);
555         if(abilityUsed_ && game_.GetTime() < abilityStartTime_ + 2.0F) {
556             float t = game_.GetTime() - abilityStartTime_;
557             int frame = (int)(t * 60.0F);
558             r.RenderAnimation(AnimationID::hand_whirl, frame, x_, y_, 3.0F, rot_, game_.GetCamera());
559         }
560     }
561 
562 protected:
563     virtual void Ability(float x, float y) {
564         abilityStartTime_ = game_.GetTime();
565     }
566 
567     int whooshCounter = 0;
568     float abilityStartTime_ = 0;
569 };
570 
571 
```

```
573 class TIKTeekkari : public Teekkari {
574 public:
575     TIKTeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y,
      rot, data) {}
576     TIKTeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
      gm::GameObjectType::teekkari_ik, x, y, rot) {}
577 protected:
578     virtual void Ability(float x, float y) {
579 
580         game_.GetAudioSystem().PlaySound(SoundID::glitch_sound);
581         game_.AddObject(std::make_unique<Effect>(game_, AnimationID::matrix_bug,
582         x_, y_, 0.0F, 1.0F, 60.0F, 0.1F));
583         SetX(x_ + 8.0F);
584         Record();
585         game_.AddObject(std::make_unique<Effect>(game_, AnimationID::matrix_bug,
586         x_, y_, 0.0F, 1.0F, 60.0F, 0.1F));
587 
588 
589     }
590 };
592 class INKUBIOTeekkari : public Teekkari {
593 public:
594     INKUBIOTeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y,
      rot, data) {}
595     INKUBIOTeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
      gm::GameObjectType::teekkari_ik, x, y, rot) {}
596 protected:
597     virtual void Ability(float x, float y) {
598         auto cow = std::make_unique<AbilityCow>(game_, x_, y_, 0.0F);
599         game_.AddObject(std::move(cow));
600     }
601 };
602 
603 
605 class KIKTeekkari : public Teekkari {
606 public:
607     KIKTeekkari(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y,
      rot, data) {}
608     KIKTeekkari(Game& game, float x, float y, float rot) : Teekkari(game,
      gm::GameObjectType::teekkari_ik, x, y, rot) {}
609 
610     virtual void Update() {
611         if(abilityUsed_ && wrenchesShot_ < 3 && game_.GetTime() - lastShotTime_ > shootingInterval_) {
612             int id = game_.AddObject(std::make_unique<AbilityWrench>(game_, x_ + targetDir.x * 0.5F, y_
      + targetDir.y * 0.5F, 0.0F));
613             AbilityWrench& wrench = (AbilityWrench&)game_.GetObject(id);
614             wrench.Impulse({250.0F * targetDir.x, 250.0F * targetDir.y});
615             lastShotTime_ = game_.GetTime();
616             wrenchesShot_++;
617         }
618         Teekkari::Update();
619     }
620 
621 protected:
622     virtual void Ability(float x, float y) {
623         auto relativeCoords =
      game_.GetScreen().GetApplication().GetRenderSystem().GetRelativeCoords({x_, y_}, game_.GetCamera());
624         targetDir = {x - relativeCoords.x, relativeCoords.y - y};
625         targetDir.Normalize();
626         mainBody_->SetLinearVelocity({0, 0});
627         headBody_->SetLinearVelocity({0, 0});
628         mainBody_->ApplyAngularImpulse(1800.0F, true);
629         armRBody_->ApplyLinearImpulseToCenter({targetDir.x * -1000.0F, targetDir.y * -1000.0F}, true);
630     }
631 
632     float shootingInterval_ = 0.2F;
633     float lastShotTime_ = 0;
634     int wrenchesShot_ = 0;
635     b2Vec2 targetDir = {0, -1};
636 };
637 
638 
639 
641 class Professor : public Teekkari {
642 public:
643     Professor(Game& game, float x, float y, float rot, gm::PersonData data) : Teekkari(game, x, y, rot,
      data) {}
644     Professor(Game& game, float x, float y, float rot) : Teekkari(game, gm::GameObjectType::teekkari_ik,
      x, y, rot) {}
645 
646     virtual void Update() {
647         updCount_++;
648         if(abilityUsed_ && GetRealTime() > abilityStartTime_ + 3.2F) {
649             if(!resumed_) {
650                 resumed_ = true;
651                 game_.ProfessorResume();
652             }
```

```
653         }
654
655         else if(abilityUsed_) {
656
657             float t = GetRealTime() - abilityStartTime_;
658             t /= 3.2F;
659
660             auto v = tVelocity_;
661             v = {v.x * 0.9F, v.y * 0.9F};
662             SetX(x_ + ph::timestep * v.x);
663             //tY_ = tY_ + ph::timestep * v.y;
664             tVelocity_ = v;
665             SetY(tY_ + sinf(t * 2 * ph::pi));
666
667             int rDeg = abs((int)rot_) % 360;
668             int rDeg2 = rDeg % 180;
669
670             if(rDeg> 5) SetRotation(rot_ + ((rDeg > 180) ? ph::timestep * rDeg2 : -ph::timestep *
    rDeg2));
671
672             for(int i = 0; i < 8; i++) {
673                 sf::Vector2f v = ph::rotateVector(1.0F + sinf(ph::pi * t) * 3.0F, 0.0F, t * 360 + i *
    45.0F);
674                 particles_.at(i)->SetPosition(x_ + v.x, y_ + v.y);
675             }
676
677         }
678
679         Teekkari::Update();
680     }
681
682 protected:
683     virtual void Ability(float x, float y) {
684         abilityStartTime_ = game_.GetTime();
685         updCount_ = 0;
686         tVelocity_ = mainBody_->GetLinearVelocity();
687         tY_ = y_;
688         game_.ProfessorPause();
689
690         SoundID sounds[] = {
691             SoundID::professor_oneliner1,
692             SoundID::professor_oneliner2,
693             SoundID::professor_oneliner3,
694             SoundID::professor_oneliner4,
695             SoundID::professor_oneliner5,
696             SoundID::professor_oneliner6,
697             SoundID::professor_oneliner7,
698             SoundID::professor_oneliner8
699         };
700
701         game_.GetAudioSystem().PlaySound(sounds[rng::RandomInt(0, 7)]);
702         game_.GetAudioSystem().PlaySound(SoundID::integral_destruction, 0.6F);
703         game_.AddObject(std::make_unique<AbilityIntegral>(game_, x_, y_, 0.0F));
704
705         for(int i = 0; i < 8; i++) {
706
707             sf::Vector2f v = ph::rotateVector(1.0F, 0.0F, i * 45.0F);
708
709             int id = game_.AddObject(std::make_unique<ProfessorParticle>(game_, x_ + v.x, y_ + v.y,
    0.0F));
710             ProfessorParticle& pp = (ProfessorParticle&)game_.GetObject(id);
711
712             pp.SetSize(1.0F);
713             pp.SetSprite((i % 2) ? SpriteID::gravity_symbols : SpriteID::math_cloud);
714             pp.SetLifeTime(3.3F);
715
716             particles_.push_back(&pp);
717         }
718
719     }
720
721     float GetRealTime() {
722         return abilityStartTime_ + updCount_ * ph::timestep;
723     }
724
725     bool resumed_ = false;
726     int updCount_ = 0;
727     float abilityStartTime_ = 0;
728
729     b2Vec2 tVelocity_ = {0, 0};
730     float tY_ = 0;
731
732     std::vector<ProfessorParticle*> particles_;
733
734
735 };
736
```

```
737
738 #endif
```

7.27 Terrain.hpp

```
1 #ifndef GAME_TERRAIN_HPP
2 #define GAME_TERRAIN_HPP
3
4 #include <gameplay/Block.hpp>
5 #include <gameplay/PhysObject.hpp>
6 #include <gameplay/GameObjectTypes.hpp>
7 #include <framework/RenderSystem.hpp>
8 #include <framework/RandomGen.hpp>
9 #include <gameplay/ParticleEffect.hpp>
10 #include <box2d/b2_circle_shape.h>
11 #include <box2d/b2_fixture.h>
12 #include <box2d/b2_world.h>
13 #include <box2d/b2_body.h>
14 #include <box2d/b2_api.h>
15 #include <unordered_map>
16 #include <limits>
17
19 class Terrain : public Block {
20 public:
22     Terrain(Game& game, float x, float y, float rot) : Block(game, gm::GameObjectType::terrain1x1, x, y,
      rot) {
23         mainBody_->SetGravityScale(0);
24         mainBody_->SetType(b2BodyType::b2_staticBody);
25         hp_ = std::numeric_limits<float>::infinity();
26     }
27
28     virtual float GetMass() const { return ph::groundMass; }
29
31     virtual void DealDamage(float damage) { }
32
33
34
35
36
37 };
38
39
40
41 #endif
```

7.28 Tnt.hpp

```
1 #ifndef GAME_TNT_HPP
2 #define GAME_TNT_HPP
3
4 #include <gameplay/Block.hpp>
5 #include <gameplay/PhysObject.hpp>
6 #include <gameplay/GameObjectTypes.hpp>
7 #include <framework/RenderSystem.hpp>
8 #include <framework/RandomGen.hpp>
9 #include <box2d/b2_circle_shape.h>
10 #include <box2d/b2_fixture.h>
11 #include <box2d/b2_world.h>
12 #include <box2d/b2_body.h>
13 #include <box2d/b2_api.h>
14
16 class Tnt : public Block {
17 public:
19     Tnt(Game& game, float x, float y, float rot) : Block(game, gm::GameObjectType::prop_tnt, x, y, rot)
      {}
20
21 protected:
23     virtual void OnDeath() {
24
25         Block::OnDeath();
26         auto objs = game_.GetObjects();
27
28         b2CircleShape circle;
29         circle.m_radius = 5.0F;
30
31         for(auto o : objs) {
32             if(o->GetGameID() != gameID_) {
33                 auto physBodies = o->GetPhysBodies();
34                 bool hit = false;
```

```
35                 for(auto p : physBodies) {
36                     if(b2TestOverlap(&circle, 0, p->GetFixtureList()[0].GetShape(), 0,
       mainBody_->GetTransform(), p->GetTransform())) {
37                         hit = true;
38                         break;
39                     }
40                 }
41                 if(hit) {
42                     PhysObject* phys = (PhysObject*)o;
43                     phys->ExplosionDamage({x_, y_}, 100 + phys->GetMass() * 5);
44                     phys->Explosion({x_, y_}, phys->GetMass() * 100.0F);
45                 }
46             }
47         }
48 
49         game_.GetAudioSystem().PlaySound(rng::RandomInt(0, 1) ? SoundID::tnt_explode1 :
       SoundID::tnt_explode2);
50         game_.AddObject(std::make_unique<Effect>(game_, AnimationID::explosion,
51         x_, y_, 0.0F, 4.0F, 60.0F, 0.35F));
52 
53 
54     }
55 };
56 
57 
58 
59 #endif
```

7.29 GameScreen.hpp

```
1 #ifndef GAME_SCREEN_HPP
2 #define GAME_SCREEN_HPP
3 
4 #include <memory>
5 #include <gameplay/Editor.hpp>
6 #include <gameplay/Level.hpp>
7 #include <gameplay/Physics.hpp>
8 #include <screens/Screen.hpp>
9 #include <screens/MainMenu.hpp>
10 #include <ui/TextLine.hpp>
11 #include <gameplay/Person.hpp>
12 #include <ui/InputElement.hpp>
13 #include <ui/DivElement.hpp>
14 class Editor;
15 
17 class GameScreen : public Screen {
18 public:
19 
21     GameScreen(Application& app, const Level& initialLevel, bool editorMode = false);
22 
23     virtual void Update();
24 
25     virtual void Render(const RenderSystem& r);
26 
27     virtual bool OnMouseDown(const sf::Mouse::Button& e, float x, float y);
28 
29     virtual bool OnMouseUp(const sf::Mouse::Button& e, float x, float y);
30 
31     virtual bool OnMouseScroll(float delta, float xw, float yh);
32 
33     virtual bool OnMouseMove(float x, float y);
34 
35     virtual bool OnKeyDown(const sf::Event::KeyEvent&);
36 
37     virtual bool OnKeyUp(const sf::Event::KeyEvent&);
38 
39     virtual ~GameScreen() = default;
40 
42     void Exit();
43 
45     void Restart();
46 
48     void OnGameCompleted(int score, int requiredMaxScore);
49 
51     void OnGameLost(const std::string& reason = "Level failed!");
52 
54     void OnScoreChange(int score);
55 
57     void UpdateProjectileList(std::vector<std::pair<SpriteID, std::string>);
58 
60     void UpdateTheoreticalMaxScore(int maxScore);
61 
62     Game& GetGame();
```

```
63
65      Editor& GetEditor();
66
67      bool IsInEditorMode() const {return editorMode_;}
68
70      bool SaveEditor();
71
75      ui::pfloat calcTopLeftButtonLeft(unsigned char buttonNumber) const;
76
80      ui::pfloat calcTopRightLabelTop(unsigned char labelNumber) const;
81
82      ui::pfloat calcTopRightLabelLeft() const;
83
84      ui::pfloat calcVictoryMessageStarTop() const;
85
86      ui::pfloat calcVictoryMessageStarLeft(char starNumber) const;
87
88      ui::pfloat calcVictoryMessageScoreTop() const;
89
90      ui::pfloat calcVictoryMessageContentLeft() const;
91
92      ui::pfloat calcVictoryMessageContentWidth() const;
93
94      ui::pfloat calcVictoryMessageNicknamePromptTop() const;
95
96      ui::pfloat calcVictoryMessageInputTop() const;
97
99      void saveScore(const std::string& name, int score);
100
101      ui::pfloat calcProjectileBarWidth() const;
102
103      ui::pfloat calcProjectileBarTop() const;
104
105      ui::pfloat calcProjectileBarBottomTop() const;
106
107      ui::pfloat calcProjectileBarBodyTop() const;
108
109      ui::pfloat calcProjectileBarBodyHeight() const;
110
111      void selectProjectileIcon(std::shared_ptr<RoundIcon> i);
112
113      void autoSelectProjectileIcon();
114
115      void unselectProjectileIcon();
116
117      ui::pfloat calcEditorPanelLeft() const;
118
119      ui::pfloat calcEditorContentWidth() const;
120
121      ui::pfloat calcEditorContentLeft() const;
122
123      ui::pfloat calcEditorDropDownTop() const;
124
125      void addDropDownContents(std::shared_ptr<TextElement> e);
126
127      ui::pfloat calcEditorMaxScoreLabelTop() const;
128
129      ui::pfloat calcEditorRequiredScoreLabelTop() const;
130
131      ui::pfloat calcEditorRequiredScoreInputTop() const;
132
133      ui::pfloat calcEditorTimeLimitLabelTop() const;
134
135      ui::pfloat calcEditorTimeLimitInputTop() const;
136
137      ui::pfloat calcEditorElementListTop() const;
138
139      ui::pfloat calcEditorElementListHeight() const;
140
141      ui::pfloat calcEditorPanelVisibilityButtonLeft() const;
142
143      void showTimeTrialOptions();
144
145      void hideTimeTrialOptions();
146
148      void setSelectedGameMode(LevelMode m);
149
150      void hideEditorPanel();
151
152      void showEditorPanel();
153
154  private:
155      const ui::pfloat topLeftButtonSpacing_ = 1 VH;
156      const ui::pfloat topLeftButtonSize_ = 4 VH;
157      const ui::pfloat topRightLabelLength_ = ui::defaultFontSize * 9;
158      const ui::pfloat topRightLabelHeigth_ = ui::defaultFontSize;
159      const ui::pfloat topRightLabelSpacing_ = 1 VH;
```

```
160     const ui::pfloat victoryMessageHeight_ = 20 VH;
161     const ui::pfloat victoryMessageWidth_ = 30 VW;
162     const ui::pfloat victoryMessageStarSize_ = 5 VH;
163     const ui::pfloat victoryMessageFontSize_ = ui::defaultFontSize;
164     const ui::pfloat victoryMessageStarSpacing_ = 2 VW;
165     const ui::pfloat projectileBarHeight_ = 50 VH;
166     const ui::pfloat projectileBarIconSize_ = 8 VH;
167     const ui::pfloat projectileBarSpacing_ = 1 VH;
168     const ui::pfloat editorPanelWidth_ = 20 VW;
169     const ui::pfloat editorPanelPadding_ = 1 VH;
170     const ui::pfloat editorPanelSpacing_ = 1 VH;
171     const ui::pfloat editorFontSize_ = ui::defaultFontSize;
172     const ui::pfloat editorElementListLineHeight_ = 4 VH;
173     const ui::pfloat editorElementListSpacing_ = 0.2 VH;
174
175     std::unique_ptr<Game> game_;
176     bool gameInitialized_ = false;
177     Level level_;
178     std::shared_ptr<TextLine> scoreLabel_;
179     std::shared_ptr<TextLine> timeLabel_;
180     std::shared_ptr<ListElement> projectileList_;
181     std::vector<std::size_t> iconIndexes_;
182     std::shared_ptr<RoundIcon> selectedIcon_;
183     bool hasSelectedIcon_ = false;
184     bool editorMode_;
185     std::shared_ptr<InputElement> editorNameInput_;
186     LevelMode selectedGameMode_ = LevelMode::normal;
187     std::shared_ptr<TextLine> editorMaxScoreLabel_;
188     std::shared_ptr<InputElement> editorRequiredScoreInput_;
189     std::shared_ptr<ListElement> editorElementList_;
190     bool timeTrial_ = false;
191     std::vector<std::shared_ptr<Element» timeTrialElements_;
192     std::shared_ptr<InputElement> editorTimeInput_;
193     std::shared_ptr<TextElement> editorGameModeDropDown_;
194     std::shared_ptr<DivElement> editorPanelDiv_;
195
199     std::shared_ptr<RoundIcon> addTopLeftButton(
200         unsigned char buttonNumber, std::function<void()> callBack, const SpriteID& sprite
201     );
202
206     std::shared_ptr<RoundIcon> addTopLeftButton(
207         unsigned char buttonNumber, const SpriteID& sprite
208     );
209
210     void addTopLeftButtons();
211
212     void addEditorTopLeftButtons();
213
214     void addTopRightLabels();
215
216     std::shared_ptr<TextLine> addTopRightLabel(unsigned char labelNumber, const std::string& text);
217
218     void addVictoryMessageStars(int score, int maxScore, std::vector<std::shared_ptr<Element»& v);
219
220     std::shared_ptr<RoundIcon> generateVictoryMessageStar(char starNumber, bool achieved);
221
222     void addVictoryMessageScore(int score, std::vector<std::shared_ptr<Element»& v);
223
224     void addVictoryMessageNicknamePrompt(std::vector<std::shared_ptr<Element»& v);
225
226     std::shared_ptr<InputElement> generateVictoryMessageInput();
227
228     void addProjectileBar();
229
230     std::shared_ptr<ColoredElement> addListTop();
231
232     std::shared_ptr<ColoredElement> addListBottom();
233
234     void addList();
235
236     void addProjectileIcon(SpriteID icon, const std::string& name);
237
238     void clearIcons();
239
240     void addEditorPanel();
241
242     void addEditorPanelBackground();
243
244     void addEditorNameInput();
245
246     void addEditorGameModeDropDown();
247
248     void addEditorMaxScoreLabel();
249
250     void addEditorRequiredScoreLabel();
251
252     void addEditorRequiredScoreInput();
```

```
253
254     void addEditorTimeLimitLabel();
255
256     void addEditorTimeLimitInput();
257
258     void addEditorElementList();
259
260     std::shared_ptr<DivElement> addEditorElementListLine(
261         SpriteID icon, const std::string& text, const std::function<void()> mouseDownHandler
262     );
263
264     void addBlocksToEditorElementList();
265
266     void addProjectilesToEditorElementList();
267
268     void addFuksiToEditorElementList();
269
270     void addEditorPanelVisibilityButton();
271 };
272
273
274 #endif
```

7.30 MainMenu.hpp

```
1 #ifndef SCREENS_MAINMENU_HPP
2 #define SCREENS_MAINMENU_HPP
3
4 #include <screens/Screen.hpp>
5 #include <ui/ListElement.hpp>
6 #include <ui/Button.hpp>
7 #include <ui/MultilineText.hpp>
8 #include <gameplay/Level.hpp>
9
11 class MainMenu : public Screen {
12 public:
13     MainMenu(Application& app);
14
15     virtual void Render(const RenderSystem&);
16
18     void SelectLevel(const Level& level, std::weak_ptr<Button> button, int id);
19
21     Level GetSelectedLevel() const {return selectedLevel_.first;};
22
23     ui::pfloat calcListWidth() const;
24
25     ui::pfloat calcListElementWidth() const;
26
27     ui::pfloat calcRightSideElementWidth() const;
28
29     ui::pfloat calcListTop() const;
30
31     ui::pfloat calcListHeight() const;
32
33     ui::pfloat calcListBottomTop() const;
34
35     ui::pfloat calcRightSideButtonTop(unsigned char buttonNumber) const;
36
37     ui::pfloat calcRightSideLeft() const;
38
39     ui::pfloat calcScoreboardMultilineTop() const;
40
41     ui::pfloat calcScoreboardMultilineHeight() const;
42
43     void deleteSelectedLevel();
44
45 private:
46     const ui::pfloat padding_ = 2 VH;
47     const ui::pfloat listPadding_ = 1 VH;
48     const ui::pfloat listSpacing_ = 1 VH;
49     const ui::pfloat spacingY_ = 1 VH;
50     const ui::pfloat spacingX_ = 1 VW;
51     const ui::pfloat buttonHeight_ = 5 VH;
52     const ui::pfloat scoreboardLeftPadding_ = 0.5 VW;
53     const sf::Color selectedLevelBackground_ = ui::highlightColor;
54     const ui::pfloat scoreboardHeaderSize_ = ui::defaultFontSize * 4;
55
56     std::shared_ptr<ListElement> list_;
57     std::shared_ptr<ColoredElement> listTop_;
58     std::shared_ptr<ColoredElement> listBottom_;
59     std::pair<Level, std::weak_ptr<Button>> selectedLevel_;
60     bool hasSelectedLevel_ = false;
61     int selectedLevelButtonListID_;
```

```
62      std::shared_ptr<MultilineText> scoreboard_;
63      std::vector<std::shared_ptr<Button» deactivatingButtons_;
64      unsigned char nofButtons_ = 0;
65
66      void generateLevels();
67
68      void addLevel(Level level);
69
70      void addRightSideButton(
71          unsigned char buttonNumber,
72          const std::string& text,
73          const std::function<void()> mouseDownHandler,
74          bool deactivating = false
75      );
76
77      void addScoreboard();
78
79      std::shared_ptr<TextElement> addScoreboardHeader();
80
81      std::shared_ptr<MultilineText> addScoreboardMultiline();
82
83      std::string generateScoreboardText(const Level& level) const;
84
85      std::shared_ptr<ListElement> addList();
86
87      std::shared_ptr<ListElement> addListBody();
88
89      std::shared_ptr<ColoredElement> addListTop();
90
91      std::shared_ptr<ColoredElement> addListBottom();
92  };
93
94  #endif
```

7.31 Screen.hpp

```
1  #ifndef SCREEN_HPP
2  #define SCREEN_HPP
3
4  #include <memory>
5  #include <vector>
6  #include <ui/Element.hpp>
7  #include <UpdateListener.hpp>
8  #include <Application.hpp>
9  #include <queue>
10 #include <ui/RoundIcon.hpp>
11 #include <ui/TextElement.hpp>
12 #include <sstream>
13 #include <exception>
14
15 #include <iostream>
16
17 class Application;
18
20 class Screen : public UpdateListener {
21 public:
22      Screen(Application& app) : app_(app) {}
23
24      virtual ~Screen() = default;
25
26      virtual void Update() {
27          //Update all elements
28          //for(const auto& e : menu_) e->Update();
29          //GameScreen overrides this
30      }
31
32      virtual void Render(const RenderSystem& r);
33
34      Application& GetApplication() const { return app_; }
35
36      virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh);
37
38      virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh);
39
40      virtual bool OnMouseMove(float xw, float yh);
41
42      virtual bool OnMouseScroll(float delta, float xw, float yh);
43
44      virtual bool OnTextEntered(const sf::Event::TextEvent&);
45
46      virtual bool OnKeyDown(const sf::Event::KeyEvent&);
47
48      virtual bool OnKeyUp(const sf::Event::KeyEvent&);
```

```
49
51      void Confirm(std::string text, const std::function<void(bool)> callBack);
52
54      void Alert(std::string text, const std::function<void()> callBack);
55      void Alert(std::string text);
56
58      void PlayClickSound() const;
59
61      void DequeueMessage();
62
63      ui::pfloat calcMessageBoxButtonTop(const ui::pfloat& messageHeight) const;
64
68      ui::pfloat calcMessageBoxButtonLeft(
69          unsigned char buttonNumber,
70          const ui::pfloat& messageWidth
71      ) const;
72
73      ui::pfloat calcConfirmTextTop() const;
74
75      ui::pfloat calcConfirmTextLeft() const;
76
77      ui::pfloat calcConfirmTextHeight() const;
78
79      ui::pfloat calcConfirmTextWidth() const;
80
81  protected:
82      const ui::pfloat messageBoxHeight_ = 15 VH;
83      const ui::pfloat messageBoxWidth_ = 30 VW;
84      const ui::pfloat messageBoxButtonSize_ = 4 VH;
85      const ui::pfloat messageBoxSpacing_ = 1 VH;
86
87      Application& app_;
88      std::vector<std::shared_ptr<Element» menu_;
89      std::queue<std::vector<std::shared_ptr<Element»> messages_;
90      float windowWidth_ = 0.0F;
91      float windowHeight_ = 0.0F;
92      std::shared_ptr<Element> focusedElement_;
93      bool hasFocusedElement_ = false;
94
95      template <typename T>
96      std::string getString(T v) const {
97          std::stringstream ss;
98          ss « v;
99          return ss.str();
100     }
101
102     int parseInt(std::string s) const {
103         int i = std::stoi(s);
104         if(getString(i) != s) throw std::invalid_argument("");
105         return i;
106     }
107
108     bool isEmpty(std::string s) const {
109         std::size_t len = s.size();
110         for(std::size_t i = 0; i < len; i++) if(s[i] != ' ') return false;
111         return true;
112     }
113
117     std::shared_ptr<RoundIcon> generateMessageBoxButton(
118         unsigned char buttonNumber,
119         const std::function<void()> callBack,
120         const SpriteID& sprite,
121         const ui::pfloat& messageHeight,
122         const ui::pfloat& messageWidth
123     );
124
125     std::shared_ptr<TextElement> generateConfirmText(const std::string& text);
126
127     void setFocusedElement(const std::shared_ptr<Element>&);
128 };
129
130
131 #endif
```

7.32 Button.hpp

```
1 #ifndef UI_BUTTON_HPP
2 #define UI_BUTTON_HPP
3
4 #include <ui/TextElement.hpp>
5
7 class Button: public TextElement{
8 public:
```

```
9       Button(
10          const ui::pfloat& top,
11          const ui::pfloat& left,
12          const ui::pfloat& height,
13          const ui::pfloat& width
14      ): TextElement(top, left, height, width){
15          defaultBackgroundColor_ = ui::buttonBackgroundColor;
16          backgroundColor_ = ui::buttonBackgroundColor;
17          textColor_ = ui::buttonTextColor;
18          align_ = ui::TextAlign::center;
19      };
20
21      Button(
22          const ui::pfloat& top,
23          const ui::pfloat& left,
24          const ui::pfloat& height,
25          const ui::pfloat& width,
26          const std::function<void()> mouseDownHandler
27      ): Button(top, left, height, width){
28          mouseDownHandler_ = mouseDownHandler;
29      };
30
32      void Deactivate(){active_ = false;};
33
35      void Activate(){active_ = true;};
36
37      void SetDeactivatedBackgroundColor(const sf::Color& c){deactivatedBackgroundColor_ = c;}
38      void SetDeactivatedBackgroundColor(){deactivatedBackgroundColor_ =
        defaultDeactivatedBackgroundColor_;}
39
40      virtual void Render(const RenderSystem&);
41
44      virtual void ExecuteOnMouseDown();
45
46      virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh);
47
48  private:
49      bool active_ = true;
50      sf::Color defaultDeactivatedBackgroundColor_ = ui::deactivatedButtonBackgroundColor;
51      sf::Color deactivatedBackgroundColor_ = ui::deactivatedButtonBackgroundColor;
52  };
53
54  #endif
```

7.33 ColoredElement.hpp

```
1  #ifndef UI_COLOREDELEMENT_HPP
2  #define UI_COLOREDELEMENT_HPP
3
4  #include <ui/Element.hpp>
5
7  class ColoredElement: public Element{
8  public:
9       ColoredElement(
10          const ui::pfloat& top,
11          const ui::pfloat& left,
12          const ui::pfloat& height,
13          const ui::pfloat& width
14      ): Element(top, left, height, width){}
15
16      void SetBackgroundColor(const sf::Color& c){backgroundColor_ = c;}
17      void SetBackgroundColor(){backgroundColor_ = defaultBackgroundColor_;}
18
19      virtual void Render(const RenderSystem& r){
20          if(cropped_) r.RenderRect(backgroundColor_, GetLeft(), GetTop(), w_, h_, cropArea_);
21          else r.RenderRect(backgroundColor_, GetLeft(), GetTop(), w_, h_);
22      }
23
24  protected:
25      sf::Color backgroundColor_ = ui::backgroundColor;
26      sf::Color defaultBackgroundColor_ = ui::backgroundColor;
27  };
28
29  #endif
```

7.34 DivElement.hpp

```
1  #ifndef UI_DIVELEMENT_HPP
2  #define UI_DIVELEMENT_HPP
```

```
3
4 #include <ui/ColoredElement.hpp>
5 #include <limits.h>
6
9 class DivElement: public ColoredElement{
10 public:
11     DivElement(
12         const ui::pfloat& top,
13         const ui::pfloat& left,
14         const ui::pfloat& height,
15         const ui::pfloat& width
16     ): ColoredElement(top, left, height, width){};
17
20     int InsertElement(std::shared_ptr<Element> element);
21
23     void RemoveElement(int id);
24
26     std::shared_ptr<Element> GetElement(int id);
27
28     const std::map<int, std::shared_ptr<Element>& GetElements() const;
29
31     void ClearElements();
32
33     virtual void SetPosition(ui::pfloat x, ui::pfloat y);
34
35     virtual void SetTop(ui::pfloat top);
36
37     virtual void SetLeft(ui::pfloat left);
38
39     virtual void SetSize(ui::pfloat w, ui::pfloat h);
40
41     virtual void SetHeight(ui::pfloat height);
42
43     virtual void SetWidth(ui::pfloat width);
44
45     virtual void OnWindowResize();
46
47     virtual void SetOffsetX(const ui::pfloat& ox);
48     virtual void SetOffsetX();
49
50     virtual void SetOffsetY(const ui::pfloat& oy);
51     virtual void SetOffsetY();
52
53     virtual void SetCropArea(const ui::CropArea& a);
54     virtual void SetCropArea();
55
57     virtual void Hide();
59     virtual void Show();
60
61 private:
62     int nextId_ = INT_MIN;
63     std::map<int, std::shared_ptr<Element>> elements_;
64
65     void updateValues();
66 };
67
68 #endif
```

7.35 Element.hpp

```
1 #ifndef ELEMENT_HPP
2 #define ELEMENT_HPP
3
4 #include <functional>
5 #include <ui/UIConstants.hpp>
6 #include <UpdateListener.hpp>
7
8
10 class Element : public UpdateListener {
11 public:
12     Element(
13         const ui::pfloat& top,
14         const ui::pfloat& left,
15         const ui::pfloat& height,
16         const ui::pfloat& width
17     ): x_(left), y_(top), w_(width), h_(height){};
18
19     virtual ~Element() = default;
20
21     virtual void Render(const RenderSystem&) = 0;
22
23     virtual void SetPosition(ui::pfloat x, ui::pfloat y) { x_ = x; y_ = y; }
24
```

```
25      virtual void SetTop(ui::pfloat top){y_ = top;}
26
28      ui::pfloat GetTopY() const {return y_;}
29
30      virtual void SetLeft(ui::pfloat left){x_ = left;}
31
33      ui::pfloat GetLeftX() const {return x_;}
34
35      virtual void SetSize(ui::pfloat w, ui::pfloat h) { w_ = w; h_ = h; }
36
37      virtual void SetHeight(ui::pfloat height){h_ = height;}
38
39      ui::pfloat GetHeight() const {return h_;}
40
41      virtual void SetWidth(ui::pfloat width){w_ = width;}
42
43      ui::pfloat GetWidth() const {return w_;}
44
46      virtual bool isInside(float xw, float yh) const;
47
50      virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh);
51
54      bool ClickSoundShouldBePlayed() const;
55
58      virtual void ExecuteOnMouseDown();
59
60      virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh);
61
62      virtual bool OnMouseMove(float xw, float yh);
63
64      virtual bool OnMouseScroll(float delta, float xw, float yh);
65
66      virtual bool OnKeyDown(const sf::Event::KeyEvent&){return false;}
67
68      virtual bool OnKeyUp(const sf::Event::KeyEvent&){return false;}
69
70      virtual bool OnTextEntered(const sf::Event::TextEvent&){return false;}
71
73      virtual void OnWindowResize();
74
75      void SetMouseDownHandler(const std::function<void()> f){mouseDownHandler_ = f;}
76      void SetMouseDownHandler(){mouseDownHandler_ = NULL;}
77
78      void SetMouseUpHandler(const std::function<void()> f){mouseUpHandler_ = f;}
79      void SetMouseUpHandler(){mouseUpHandler_ = NULL;}
80
81      void SetMouseEnterHandler(const std::function<void()> f){mouseEnterHandler_ = f;}
82      void SetMOuseEnterHandler(){mouseEnterHandler_ = NULL;}
83
84      void SetMouseLeaveHandler(const std::function<void()> f){mouseLeaveHandler_ = f;}
85      void SetMouseLeaveHandler(){mouseLeaveHandler_ = NULL;}
86
90      void SetMouseScrollHandler(const std::function<void(float delta)> f){mouseScrollHandler_ = f;}
91      void SetMouseScrollHandler(){mouseScrollHandler_ = NULL;}
92
93      void SetFocusChangeHandler(const std::function<void(bool focused)> f){focusChangeHandler_ = f;}
94      void SetFocusChangeHandler(){focusChangeHandler_ = NULL;}
95
96      void SetWindowResizeHandler(const std::function<void()> f){windowResizeHandler_ = f;}
97      void SetWindowResizeHandler(){windowResizeHandler_ = NULL;}
98
100      virtual void Blur();
101
103      virtual void Focus();
104
105      virtual void SetOffsetX(const ui::pfloat& ox){offsetX_ = ox;}
106      virtual void SetOffsetX(){offsetX_ = 0 VW;}
107
108      virtual void SetOffsetY(const ui::pfloat& oy){offsetY_ = oy;}
109      virtual void SetOffsetY(){offsetY_ = 0 VH;}
110
112      virtual void SetCropArea(const ui::CropArea& a){
113          cropArea_ = a;
114          cropped_ = true;
115      }
116      virtual void SetCropArea(){cropped_ = false;}
117
118      bool IsCropped() const {return cropped_;}
119
120      ui::CropArea GetCropArea() const {return cropArea_;}
121
123      ui::pfloat toVH(const ui::pfloat&) const;
125      ui::pfloat toVW(const ui::pfloat&) const;
126
128      ui::pfloat GetTop() const;
130      ui::pfloat GetLeft() const;
131
```

```
133     void SetFocusCapture(bool b){captureFocus_ = b;}
134
137     void SetTitle(const std::string& s);
138
139     bool IsVisible() const;
140     virtual void Hide();
141     virtual void Show();
142
143 protected:
144
145     ui::pfloat x_;
146     ui::pfloat y_;
147     ui::pfloat w_;
148     ui::pfloat h_;
149     ui::pfloat offsetX_ = 0 VW;
150     ui::pfloat offsetY_ = 0 VH;
151     bool captureFocus_ = false;
152     bool visible_ = true;
153
154     std::function<void()> mouseDownHandler_ = NULL;
155     std::function<void()> mouseUpHandler_ = NULL;
156     std::function<void()> mouseEnterHandler_ = NULL;
157     std::function<void()> mouseLeaveHandler_ = NULL;
158     std::function<void(float delta)> mouseScrollHandler_ = NULL;
159     std::function<void(bool focused)> focusChangeHandler_ = NULL;
160     std::function<void()> windowResizeHandler_ = NULL;
161
162     bool mouseIn_ = false;
163     bool focused_ = false;
164
165     bool cropped_ = false;
166     ui::CropArea cropArea_;
167
168     std::string title_ = "";
169     ui::pfloat titleFontSize_ = ui::defaultFontSize;
170     ui::pfloat titleX_;
171     ui::pfloat titleY_;
172     ui::pfloat titleW_ = 1 VW;
173     bool renderTitle = false;
174
175     bool isInsideCropArea(float xvw, float yvh) const;
176
177     void RenderTitle(const RenderSystem& r);
178 };
179
180
181
182 #endif
```

7.36 InputElement.hpp

```
1 #ifndef UI_INPUTELEMENT_HPP
2 #define UI_INPUTELEMENT_HPP
3
4 #include <ui/ColoredElement.hpp>
5
7 class InputElement: public ColoredElement{
8 public:
9     InputElement(
10         const ui::pfloat& top,
11         const ui::pfloat& left,
12         const ui::pfloat& height,
13         const ui::pfloat& width
14     ): ColoredElement(top, left, height, width){
15         captureFocus_ = true;
16         defaultBackgroundColor_ = ui::inputBackgroundColor;
17         backgroundColor_ = ui::inputBackgroundColor;
18         updateInputArea();
19     };
20
21     virtual void Render(const RenderSystem&);
22
24     void SetText(const std::string&);
25
27     std::string GetText() const;
28
29     virtual bool OnKeyDown(const sf::Event::KeyEvent&);
30
31     virtual bool OnTextEntered(const sf::Event::TextEvent&);
32
33     void SetTextColor(const sf::Color& c){textColor_ = c;}
34     void SetTextColor(){textColor_ = ui::textColor;}
35
```

```
36      void SetFont(FontID f);
37
38      void SetFontSize(const ui::pfloat& s);
39
40      ui::pfloat GetFontSize(){return fontSize_;}
41
42      virtual void SetPosition(ui::pfloat x, ui::pfloat y);
43
44      virtual void SetTop(ui::pfloat top);
45
46      virtual void SetLeft(ui::pfloat left);
47
48      virtual void SetSize(ui::pfloat w, ui::pfloat h);
49
50      virtual void SetHeight(ui::pfloat height);
51
52      virtual void SetWidth(ui::pfloat width);
53
54      virtual void OnWindowResize();
55
56      virtual void SetOffsetX(const ui::pfloat& ox);
57      virtual void SetOffsetX();
58
59      virtual void SetOffsetY(const ui::pfloat& oy);
60      virtual void SetOffsetY();
61
62      virtual void SetCropArea(const ui::CropArea& a);
63      virtual void SetCropArea();
64
65      //virtual void Blur();
66
67      //virtual void Focus();
68
69  private:
70      std::string value_ = "";
71      size_t caretPos_ = 0;
72      ui::pfloat textOffset_ = 0 VH;
73      ui::pfloat caretOffset_ = 0 VH;
74      bool modified_ = true;
75      ui::pfloat fontSize_ = ui::defaultFontSize;
76      FontID font_ = ui::defaultMonospaceFont;
77      sf::Color textColor_ = ui::textColor;
78      ui::CropArea inputArea_;
79      sf::Color caretColor_ = ui::inputCaretColor;
80      ui::pfloat caretWidth_ = ui::defaultFontSize / 8;
81
82      void write(char);
83
84      void backspace();
85
86      void moveCaretRight();
87
88      void moveCaretLeft();
89
93      char getChar(sf::Uint32) const;
94
95      void updateInputRenderingValues(const RenderSystem& r);
96
97      void updateInputArea();
98  };
99
100 #endif
```

7.37 ListElement.hpp

```
1  #ifndef UI_LIST_HPP
2  #define UI_LIST_HPP
3
4  #include <ui/ColoredElement.hpp>
5  #include <memory>
6  #include <limits.h>
7
10 class ListElement: public ColoredElement{
11 public:
12      ListElement(
13          const ui::pfloat& top,
14          const ui::pfloat& left,
15          const ui::pfloat& height,
16          const ui::pfloat& width
17      ): ColoredElement(top, left, height, width){}
18
19      virtual bool OnMouseMove(float xw, float yh);
20
```

```
21      virtual bool OnMouseScroll(float delta, float xw, float yh);
22
25      int InsertElement(std::shared_ptr<Element> element);
26
28      void RemoveElement(int id);
29
31      std::shared_ptr<Element> GetElement(int id);
32
34      void SetSpacing(const ui::pfloat&);
35
39      const std::map<int, std::shared_ptr<Element»& GetElements() const;
40
42      void ClearElements();
43
44      virtual void SetPosition(ui::pfloat x, ui::pfloat y);
45
46      virtual void SetTop(ui::pfloat top);
47
48      virtual void SetLeft(ui::pfloat left);
49
50      virtual void SetSize(ui::pfloat w, ui::pfloat h);
51
52      virtual void SetHeight(ui::pfloat height);
53
54      virtual void SetWidth(ui::pfloat width);
55
56      virtual void OnWindowResize();
57
58      virtual void SetOffsetX(const ui::pfloat& ox);
59      virtual void SetOffsetX();
60
61      virtual void SetOffsetY(const ui::pfloat& oy);
62      virtual void SetOffsetY();
63
64      virtual void SetCropArea(const ui::CropArea& a);
65      virtual void SetCropArea();
66
67      virtual void Hide();
68      virtual void Show();
69
70  private:
71      int nextId_ = INT_MIN;
72      std::map<int, std::shared_ptr<Element» elements_;
73      ui::pfloat scrollOffset_;
74      ui::pfloat spacing_ = 1 VH;
75      float scrollMultiplier_ = 5000.0;
76      float lastMouseX_ = 0;
77      float lastMouseY_ = 0;
78
79      void updateValues();
80
81      ui::CropArea calcCropArea() const;
82
83      void updateScrollOffset(float delta, float xw, float yh);
84  };
85
86  #endif
```

7.38 MessageBox.hpp

```
1  #ifndef UI_MESSAGEBOX_HPP
2  #define UI_MESSAGEBOX_HPP
3
4  #include <ui/ColoredElement.hpp>
5
7  class MessageBox: public ColoredElement{
8  public:
9      MessageBox(const ui::pfloat& height, const ui::pfloat& width): ColoredElement(
10         (50 - ui::toVHFloat(height) / 2) VH, (50 - ui::toVWFloat(width) / 2) VW, height, width
11     ){}
12
13     virtual void Render(const RenderSystem&);
14
15     virtual bool OnKeyDown(const sf::Event::KeyEvent&) {return true;}
16
17     virtual bool OnKeyUp(const sf::Event::KeyEvent&) {return true;}
18
19     virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh);
20
21     virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh);
22
23     virtual bool OnMouseScroll(float delta, float xw, float yh);
24
```

```
25     virtual bool OnTextEntered(const sf::Event::TextEvent&){return true;}
26 };
27
28 #endif
```

7.39 MultilineText.hpp

```
1 #ifndef UI_MULTILINETEXT_HPP
2 #define UI_MULTILINETEXT_HPP
3
4 #include <ui/TextElement.hpp>
5
7 class MultilineText: public TextElement{
8 public:
9     MultilineText(
10         const ui::pfloat& top,
11         const ui::pfloat& left,
12         const ui::pfloat& height,
13         const ui::pfloat& width
14     ): TextElement(top, left, height, width){}
15
16     virtual void SetText(const std::string& s);
17
18     virtual void Render(const RenderSystem& r);
19
21     void SetRelativeLineSpacing(const ui::pfloat& s);
22
24     void SetAbsoluteLineSpacing(float s);
25
27     ui::pfloat GetLineSpacing();
28
29 private:
30     ui::pfloat relativeLineSpacing_ = 0 VH;
31     float absoluteLineSpacing_ = ui::defaultAbsoluteFontSize / 4;
32     bool useRelativeLineSpacing_ = false;
33     std::vector<std::string> lines_;
34 };
35
36 #endif
```

7.40 RoundElement.hpp

```
1 #ifndef UI_ROUNDELEMENT_HPP
2 #define UI_ROUNDELEMENT_HPP
3
4 #include <ui/Element.hpp>
5
7 class RoundElement: public Element{
8 public:
9     RoundElement(
10         const ui::pfloat& top,
11         const ui::pfloat& left,
12         const ui::pfloat& radius
13     ): Element(top, left, radius * 2, radius * 2), r_(radius){};
14
15     virtual bool isInside(float xw, float yh) const;
16
17 protected:
18     ui::pfloat r_;
19
20     float getCenterVHFloatX() const;
21     float getCenterVHFloatX(float rvh) const;
22
23     float getCenterVHFloatY() const;
24     float getCenterVHFloatY(float rvh) const;
25
26     float distance(float x1, float y1, float x2, float y2) const;
27 };
28
29 #endif
```

7.41 RoundIcon.hpp

```
1 #ifndef UI_ROUNDICON_HPP
2 #define UI_ROUNDICON_HPP
```

```
3
4 #include <ui/RoundElement.hpp>
5
7 class RoundIcon: public RoundElement{
8 public:
9     RoundIcon(
10         const ui::pfloat& top,
11         const ui::pfloat& left,
12         const ui::pfloat& radius,
13         const SpriteID& icon
14     ): RoundElement(top, left, radius), icon_(icon){}
15
16     virtual void Render(const RenderSystem& r);
17
18     void SetIcon(const SpriteID& icon){icon_ = icon;}
19
20     SpriteID GetIcon(){return icon_;}
21
23     void Select();
24
26     void Unselect();
27
29     void SetBorderThickness(const ui::pfloat&);
30
31 private:
32     SpriteID icon_;
33     bool selected_ = false;
34     ui::pfloat borderThickness_ = 0.5 VH;
35 };
36
37 #endif
```

7.42 TextElement.hpp

```
1 #ifndef UI_TEXTELEMENT_HPP
2 #define UI_TEXTELEMENT_HPP
3
4 #include <ui/ColoredElement.hpp>
5
7 class TextElement: public ColoredElement{
8 public:
9     TextElement(
10         const ui::pfloat& top,
11         const ui::pfloat& left,
12         const ui::pfloat& height,
13         const ui::pfloat& width
14     ): ColoredElement(top, left, height, width){}
15
16     virtual void SetText(const std::string& s){text_ = s;}
17
18     void SetTextColor(const sf::Color& c){textColor_ = c;}
19     void SetTextColor(){textColor_ = ui::textColor;}
20
21     void SetFont(FontID f){font_ = f;}
22
23     void SetTextAlign(const ui::TextAlign& a){align_ = a;}
24
25     void SetRelativeFontSize(const ui::pfloat& s);
26
27     void SetAbsoluteFontSize(float s);
28
29     virtual void Render(const RenderSystem&);
30
31     ui::pfloat GetFontSize();
32
33 protected:
34     std::string text_ = "";
35     sf::Color textColor_ = ui::textColor;
36     FontID font_ = ui::defaultFont;
37     ui::TextAlign align_ = ui::TextAlign::left;
38
39 private:
40     ui::pfloat relativeFontSize_ = ui::defaultFontSize;
41     bool useRelativeFontSize_ = false;
42     float absoluteFontSize_ = ui::defaultAbsoluteFontSize;
43 };
44
45 #endif
```

7.43 TextLine.hpp

```
1 #ifndef UI_TEXTLINE_HPP
2 #define UI_TEXTLINE_HPP
3
4 #include <ui/TextElement.hpp>
6 class TextLine: public TextElement{
7 public:
8     TextLine(
9         const ui::pfloat& top,
10         const ui::pfloat& left,
11         const ui::pfloat& height,
12         const ui::pfloat& width,
13         const std::string& text
14     ): TextElement(top, left, height, width){
15         text_ = text;
16         backgroundColor_ = {0, 0, 0, 0};
17     }
18
19     virtual void Render(const RenderSystem&);
20 };
21
22 #endif
```

7.44 UIConstants.hpp

```
1 #ifndef UI_CONSTANTS_HPP
2 #define UI_CONSTANTS_HPP
3
4 #include <string>
5 #include <framework/Resources.hpp>
6 #include <SFML/Graphics/Color.hpp>
7
8 // A file defining the color scheme and style of the program
11
50 #define VW % ui::pfloat(1, ui::pfloat::P::vw)
51 #define VH % ui::pfloat(1, ui::pfloat::P::vh)
53
54
55
57 namespace ui {
58
60     struct pfloat {
61     public:
62         enum P { vh, vw };
63         pfloat() : f(0), p(P::vw) {}
64         pfloat(const float& ff, P pp) { f = ff; p = pp; }
65         operator float() const { return f; }
66         pfloat operator-() const { return { -f, p }; }
67         pfloat& operator=(const pfloat& pf) { f = pf.f; p = pf.p; return *this; }
68         pfloat& operator*=(const float& ff) { f *= ff; return *this; }
69         pfloat& operator/=(const float& ff) { f /= ff; return *this; }
70         pfloat& operator+=(const float& ff) { f += ff; return *this; }
71         pfloat& operator-=(const float& ff) { f -= ff; return *this; }
72
73         float f;
74         P p;
75     };
76     inline ui::pfloat operator%(const float& ff, const ui::pfloat& pp) { return { ff, pp.p }; }
77     // This for clang
78     inline ui::pfloat operator%(const int& ff, const ui::pfloat& pp) { return { (float) ff, pp.p }; }
79     inline ui::pfloat operator%(const double& ff, const ui::pfloat& pp) { return { (float) ff, pp.p }; }
80
81     inline ui::pfloat operator*(const float& ff, const ui::pfloat& pp){return {pp.f * ff, pp.p};}
82     inline ui::pfloat operator*(const int& ff, const ui::pfloat& pp){return {pp.f * (float)ff, pp.p};}
83     inline ui::pfloat operator*(const double& ff, const ui::pfloat& pp){return {pp.f * (float)ff, pp.p};}
84
85     inline ui::pfloat operator/(const float& ff, const ui::pfloat& pp){return {pp.f / ff, pp.p};}
86     inline ui::pfloat operator/(const int& ff, const ui::pfloat& pp){return {pp.f / (float)ff, pp.p};}
87     inline ui::pfloat operator/(const double& ff, const ui::pfloat& pp){return {pp.f / (float)ff, pp.p};}
88
89     inline ui::pfloat operator*(const ui::pfloat& pp, const float& ff){return {pp.f * ff, pp.p};}
90     inline ui::pfloat operator*(const ui::pfloat& pp, const int& ff){return {pp.f * (float)ff, pp.p};}
91     inline ui::pfloat operator*(const ui::pfloat& pp, const double& ff){return {pp.f * (float)ff, pp.p};}
92
93     inline ui::pfloat operator/(const ui::pfloat& pp, const float& ff){return {pp.f / ff, pp.p};}
94     inline ui::pfloat operator/(const ui::pfloat& pp, const int& ff){return {pp.f / (float)ff, pp.p};}
95     inline ui::pfloat operator/(const ui::pfloat& pp, const double& ff){return {pp.f / (float)ff, pp.p};}
96
97
99     struct CropArea {
100         CropArea() {}
101         CropArea(const pfloat& t, const pfloat& l, const pfloat& h, const pfloat& w) :
```

```
102         top(t), left(l), height(h), width(w) {}
103         pfloat top = 0 VH;
104         pfloat left = 0 VW;
105         pfloat height = 100 VH;
106         pfloat width = 100 VW;
107     };
108 
109     const std::string appName = "AngryTeekkari";
110     const std::string appVersion = "beta 3.7";
111     const unsigned int appMinWidth = 400; //Currently unused
112     const unsigned int appMinHeight = 400;
113     const unsigned int targetFramerate = 180;
114     const float targetFrametime = 1.0F / targetFramerate;
115 
116     enum TextAlign { left, center, right };
117     const sf::Color textColor = {0, 0, 0};
118     const sf::Color buttonTextColor = {0, 0, 0};
119     const sf::Color backgroundColor = {255, 255, 255};
120     const sf::Color backgroundColor2 = {221, 221, 221};
121     const sf::Color buttonBackgroundColor = {204, 204, 204};
122     const sf::Color messageBoxBackgroundColor = {0, 0, 0, 100};
123     const sf::Color messageBoxColor = {255, 255, 255};
124     const sf::Color highlightColor = {200, 200, 255};
125     const sf::Color deactivatedButtonBackgroundColor = {150, 150, 150};
126     const sf::Color inputBackgroundColor = {204, 204, 204};
127     const sf::Color inputCaretColor = {0, 0, 0};
128 
129     /*const sf::Color textColor = {255, 255, 255};
130     const sf::Color buttonTextColor = {255, 255, 255};
131     //const sf::Color backgroundColor = {0, 68, 66};
132     const sf::Color backgroundColor = {0, 25, 25};
133     const sf::Color backgroundColor2 = {27, 153, 150, 127};
134     //const sf::Color backgroundColor2 = {29, 156, 113, 127};
135     const sf::Color buttonBackgroundColor = {50, 219, 214};
136     const sf::Color messageBoxBackgroundColor = {0, 0, 0, 150};
137     const sf::Color messageBoxColor = {0, 68, 66};
138     const sf::Color highlightColor = {127, 255, 212};
139     const sf::Color deactivatedButtonBackgroundColor = {93, 176, 174};
140     const sf::Color inputBackgroundColor = {0, 119, 116};
141     const sf::Color inputCaretColor = {255, 255, 255};*/
142 
143     const FontID defaultFont = FontID::source_serif;
144     const FontID defaultMonospaceFont = FontID::consolas;
145 
146     const float defaultAbsoluteFontSize = 16.0F;
147     const ui::pfloat defaultFontSize = (100 * defaultAbsoluteFontSize / 1080) VH;
148 
149 
150     //Don't touch this unless u are Application
151     //plz
152     extern float windowWidth;
153     extern float windowHeight;
154     extern float aspectRatio;
155 
156 
158     float toVHFloat(const pfloat& p);
160     float toVWFloat(const pfloat& p);
161 
163     CropArea combineCropAreas(const CropArea& a, const CropArea& b);
164 
165 }
166 
167 #endif
```

7.45 UpdateListener.hpp

```
1 #ifndef UPDATE_LISTENER_HPP
2 #define UPDATE_LISTENER_HPP
3 
4 #include <framework/RenderSystem.hpp>
5 #include <SFML/Window/Event.hpp>
6 
9 class UpdateListener {
10 public:
11 
12     virtual void Update() {}
13     virtual void Render(const RenderSystem&) {}
14 
15     virtual bool OnKeyDown(const sf::Event::KeyEvent&) { return false; }
16     virtual bool OnKeyUp(const sf::Event::KeyEvent&) { return false; }
17 
18     virtual bool OnMouseDown(const sf::Mouse::Button& button, float xw, float yh) { return false; }
19     virtual bool OnMouseUp(const sf::Mouse::Button& button, float xw, float yh) { return false; }
```

```
20     virtual bool OnMouseMove(float xw, float yh) { return false; }
21
22     virtual bool OnMouseScroll(float delta, float xw, float yh) { return false; }
23
24     virtual bool OnTextEntered(const sf::Event::TextEvent&) { return false; }
25
26 protected:
27     UpdateListener() {}
28 };
29
30
31 #endif
```