Chipmunk Based 2-D Game

Detailed Design: Module Interface Specification

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

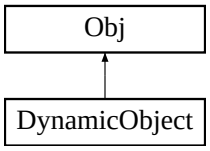
Chapter 3

Class Documentation

3.1 DynamicObject Class Reference

`#include <Obj.h>`

Inheritance diagram for DynamicObject:



Public Member Functions

- DynamicObject ()

  DynamicObject constructor.
- DynamicObject (cpSpace ∗space, glm::vec2 pos, float mass, float scale, float elast, float fric, ObjGPUData ∗gpuData, int type, bool noRotation=false)

  DynamicObject constructor.

Additional Inherited Members

3.1.1 Detailed Description

The DynamicObject class is derived from the Obj class. This type of object is subject to all physics calculations.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DynamicObject::DynamicObject (cpSpace ∗ *space*, glm::vec2 *pos*, float *mass*, float *scale*, float *elast*, float *fric*, ObjGPUData ∗ *gpuData*, int *type*, bool *noRotation* = `false`)

DynamicObject constructor.

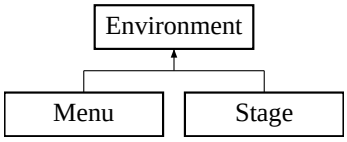
Parameters

space	Chipmunk 2D space to attach object to
pos	Coordinates of the initial center of the object
mass	Mass of the object
scale	Scalar factor by which to scale the object during GPU rendering
elast	Elasticity of the object
fric	Friction factor of the object
gpuData	Pointer to the gpu data associated with the object
type	Type of object (different values affect collision routines)
noRotation	Flag to ignore angular momentum in the physics calculations (default = false)

3.2 Environment Class Reference

`#include <Environment.h>`

Inheritance diagram for Environment:



Public Member Functions

- virtual void updateEnvironment (double dt)=0

  Pure virtual function (i.e. defined by derived classes) for updating the environment.
- virtual void drawEnvironment ()=0

  Pure virtual function (i.e. defined by derived classes) for drawing the environment.
- void changeShader (Shader ∗nextShader)

  Changes the shader program.
- virtual void processKB (int key, int scancode, int action, int mods)=0

  Pure virtual function (i.e. defined by derived classes) for keyboard input processing.
- virtual void processContinuousInput ()=0

  Pure virtual function (i.e. defined by derived classes) for continuous input processing.
- virtual void processMousePosition (float xpos, float ypos)=0

  Pure virtual function (i.e. defined by derived classes) for mouse position change processing.
- virtual void processMouseClick (int button, int action, int mods, float winX, float winY)=0

  Pure virtual function (i.e. defined by derived classes) for mouse input processing.
- void updateProjection (glm::mat4 newProjection)

  Updates the projection matrix.

Public Attributes

- int keyStates [GLFW_KEY_LAST] = {0}

  Array that keeps track of keyboard keys currently pressed down.

Protected Attributes

- std::map< std::string, ObjGPUData * > gpuMap

  Stored object data used by the GPU (i.e. meshes/texture mappings/etc.).
- std::map< std::string, Sound * > soundMap

  Stored sound data.
- std::map< std::string, Shader * > shaderMap

  Stored shaders.
- glm::mat4 mat_Projection

  Projection matrix.
- glm::mat4 mat_View

  View matrix.
- Shader * currentShader

  Pointer to the shader that is currently bound.
- float mouseX

  Current position of the mouse x-coordinate.
- float mouseY

  Current position of the mouse y-coordinate.

3.2.1 Detailed Description

The Environment class is an abstract class that holds information about the current game state and provides function calls to update and draw the game to the screen. The Stage and Menu classes are derived from this class.

3.2.2 Member Function Documentation

3.2.2.1 void Environment::changeShader (Shader * *nextShader*)

Changes the shader program.

Parameters

nextShader	New shader to be bound.

3.2.2.2 virtual void Environment::processKB (int *key*, int *scancode*, int *action*, int *mods*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for keyboard input processing.

Parameters

key	Key to which the action corresponds.
scancode	System specific key code.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implemented in Menu, and Stage.

3.2.2.3 virtual void Environment::processMouseClick (int *button*, int *action*, int *mods*, float *winX*, float *winY*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for mouse input processing.

Parameters

button	Mouse button to which the action corresponds.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)
winX	Mouse cursor x-position
winY	Mouse cursor y-position

Implemented in Menu, and Stage.

3.2.2.4 virtual void Environment::processMousePosition (float *xpos*, float *ypos*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for mouse position change processing.

Parameters

xpos	Mouse cursor x-position
ypos	Mouse cursor y-position

Implemented in Menu, and Stage.

3.2.2.5 virtual void Environment::updateEnvironment (double *dt*) [pure virtual]

Pure virtual function (i.e. defined by derived classes) for updating the environment.

Parameters

dt	Time step to be used for the update (time since last update) in milliseconds.

Implemented in Menu, and Stage.

3.2.2.6 void Environment::updateProjection (glm::mat4 *newProjection*)

Updates the projection matrix.

Parameters

newProjection	New matrix to replace previous.

3.3 Game Class Reference

`#include <Game.h>`

Public Member Functions

- Game ()

  Game class constructor.
- ∼Game ()

  Game class destructor.
- void run ()

  Runs the game until the application is terminated (infinite loop)
- void framebuffer_size_callback (GLFWwindow ∗, int, int)

  GLFW window resize callback.
- void key_callback (GLFWwindow ∗, int, int, int, int)

  GLFW keyboard input callback.
- void mouse_pos_callback (GLFWwindow ∗, float, float)

  GLFW mouse position change callback.
- void mouse_button_callback (GLFWwindow ∗, int, int, int)

  GLFW mouse button input callback.

Private Attributes

- GLFWwindow ∗ window

  Reference to the GLFWwindow (the window)
- Environment ∗ env

  Reference to the current Environment.
- double timeLast

  Last time that was polled; used for framerate control.
- double timeElapsed

  Time elapsed since last polling of time; used for framerate control.
- float winX

  Stores x-coordinate maximum of the window.
- float winY

  Stores y-coordinate maximum of the window.

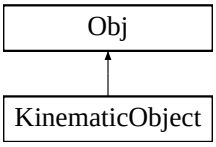
3.3.1 Detailed Description

The Game class is a representation of the game on the highest level. It handles all exchanges between the user and the game code. It keeps a reference to the game window as well as the current environment of the game (main menu, stage, etc) and acts as a bridge between the two. User inputs are intercepted through GLFW callbacks in this class and passed on for processing by the current game environment. This class is also responsible for "running" the game and sends requests for the game environment to be updated and drawn to the screen at regular intervals (framerate is controlled).

3.4 KinematicObject Class Reference

`#include <Obj.h>`

Inheritance diagram for KinematicObject:



Public Member Functions

- KinematicObject (cpSpace *space, cpVect p1, cpVect p2, ObjGPUData *gpuData, int type)
  DynamicObject constructor.

Additional Inherited Members

3.4.1 Detailed Description

The KinematicObject class is derived from the Obj class. This type of object has features of both static objects and dynamic objects and is generally used for moving platforms.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 KinematicObject::KinematicObject (cpSpace * *space*, cpVect *p1*, cpVect *p2*, ObjGPUData * *gpuData*, int *type*)

DynamicObject constructor.

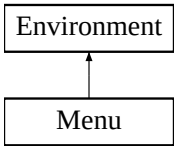
Parameters

space	Chipmunk 2D space to attach object to
p1	Bottom left coordinate of bounding box
p2	Upper right coordinate of bounding box
gpuData	Pointer to the gpu data associated with the object
type	Type of object (different values affect collision routines)

3.5 Menu Class Reference

`#include <Environment.h>`

Inheritance diagram for Menu:



Public Member Functions

- Menu ()

  Menu constructor.
- ~Menu ()

  Menu destructor.
- void updateEnvironment (double dt)

  Function for updating the environment.
- void drawEnvironment ()

  Function for drawing the environment.
- void processKB (int key, int scancode, int action, int mods)

  Function for keyboard input processing.
- void processContinuousInput ()

  Function for continuous input processing.
- void processMousePosition (float xpos, float ypos)

  Function for mouse position change processing.
- void processMouseClick (int button, int action, int mods, float winX, float winY)

  Function for mouse input processing.

Additional Inherited Members

3.5.1 Detailed Description

The Menu class is derived from the Environment class and holds and handles changes to the game state when the user is not playing a stage (i.e. is in a menu of some kind).

3.5.2 Member Function Documentation

3.5.2.1 void Menu::processKB (int *key*, int *scancode*, int *action*, int *mods*) [virtual]

Function for keyboard input processing.

Parameters

key	Key to which the action corresponds.
scancode	System specific key code.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implements Environment.

3.5.2.2 void Menu::processMouseClick (int *button*, int *action*, int *mods*, float *winX*, float *winY*) [virtual]

Function for mouse input processing.

Parameters

button	Mouse button to which the action corresponds.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)
winX	Mouse cursor x-position
winY	Mouse cursor y-position

Implements Environment.

3.5.2.3 void Menu::processMousePosition (float *xpos*, float *ypos*) [virtual]

Function for mouse position change processing.

Parameters

xpos	Mouse cursor x-position
ypos	Mouse cursor y-position

Implements Environment.

3.5.2.4 void Menu::updateEnvironment (double *dt*) [virtual]

Function for updating the environment.

Parameters

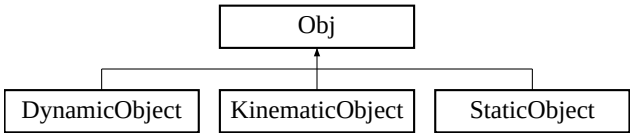
dt	Time step to be used for the update (time since last update) in milliseconds.

Implements Environment.

3.6 Obj Class Reference

`#include <Obj.h>`

Inheritance diagram for Obj:



Public Attributes

- cpBody * body

  Pointer to Chipmunk 2D body associated with the object.
- cpShape * shape

  Pointer to Chipmunk 2D shape associated with the object.
- ObjGPUData * gpuData

  Pointer to the gpu data associated with the object.
- float height

  Height of the object.
- float width

  Width of the object.
- bool draw

  Flag for whether the object should be drawn or not.

3.6.1 Detailed Description

The Obj class acts as a base class for static, dynamic, and kinematic objects. It holds the physics data (Chipmunk 2D) and gpu data of an object.

3.7 ObjGPUData Class Reference

```
#include <ObjGPUData.h>
```

Classes

- class **Material**

  Class for storing material information loaded from .mtl file.

Public Types

- enum **dataType** : short
- enum **mtlDataType** : short

Public Member Functions

- ObjGPUData (char *objFile, float angle=0.0f)

  ObjGPUData constructor.

Public Attributes

- std::vector< glm::vec3 > vList
  Stores vertex coordinates loaded from .obj file.
- std::vector< glm::vec2 > vTextureList
  Stores texture coordinates loaded from .obj file.
- std::vector< glm::vec3 > vNormalList
  Stores vertex normals loaded from .obj file.
- std::vector< GLuint > fList
  Stores faces loaded from .obj file.
- std::vector< int > materialIndices
  Marks divisions of different materials given in the .obj file (and defined in .mtl file)
- std::vector< Material > materials
  Stores material information loaded from .mtl file.
- GLuint vertexArrayObj
  Name to bind the vertex array object.
- GLuint elementBuffer
  Name to bind the element buffer object.
- GLuint vertexBuffer
  Name to bind the vertex buffer object.
- GLuint textureBuffer
  Name to bind the texture coordinate buffer object.
- GLuint normalBuffer
  Name to bind the vertex normal buffer object.
- glm::mat4 unitScale
  Scaling factor to adjust object to size 1.0 in y-axis (height)
- glm::mat4 rotation
  Rotation about y-axis to adjust objects initial rotational centering (if required: this is what the optional constructor argument sets)
- float whRatio
  Ratio of maximum x-axis vertex separation (width) to maximum y-axis vertex separation (height)

Private Member Functions

- void loadObject (char *fileName)
  Parses .obj and .mtl files and stores the data.
- dataType getDataType (std::string dataTypeString)
  Converts string to .obj file datatype (dataType).
- mtlDataType getMtlDataType (std::string dataTypeString)
  Converts string to .mtl file datatype (mtlDataType).

3.7.1 Detailed Description

The ObjGPUData class loads and stores data used by the GPU to render objects. Each object is defined by three files which are loaded by this class: an object file (.obj) which contains information about vertices, faces, normals, and texture coordinates; a material file (.mtl) which contains texture and lighting information; and an image file(s) (.dds) which contain the texture images.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 ObjGPUData::ObjGPUData (char * *objFile*, float *angle* = `0.0f`)

ObjGPUData constructor.

Parameters

objFile	Object and material file path (these should have the same name) without extension
angle	Initial y-axis rotation in radians (optional: default 0.0)

3.7.3 Member Function Documentation

3.7.3.1 ObjGPUData::dataType ObjGPUData::getDataType (std::string *dataTypeString*) [private]

Converts string to .obj file datatype (dataType).

Parameters

dataTypeString	The string to be converted.

Returns

Equivalent dataType value.

3.7.3.2 ObjGPUData::mtlDataType ObjGPUData::getMtlDataType (std::string *dataTypeString*) [private]

Converts string to .mtl file datatype (mtlDataType).

Parameters

dataTypeString	The string to be converted.

Returns

Equivalent mtlDataType value.

3.7.3.3 void ObjGPUData::loadObject (char * *fileName*) [private]

Parses .obj and .mtl files and stores the data.

Parameters

fileName	Name (without extension) of the .obj and .mtl files

3.8 Shader Class Reference

```
#include <Shader.h>
```

Public Member Functions

- Shader (const char *vShader, const char *fShader)

  Shader constructor.

Public Attributes

- GLuint shaderProgram

  Name to bind the shader program.
- std::map< std::string, GLuint > uniformIDMap

  Stores names which bind the shader uniform IDs.

3.8.1 Detailed Description

The Shader class is used to build shaders. All of the names/identifiers required to bind/use the shader afterwards are stored.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Shader::Shader (const char * *vShader*, const char * *fShader*)

Shader constructor.

Parameters

vShader	Vertex shader path
fShader	Fragment shader path

3.9 Sound Class Reference

`#include <Sound.h>`

Public Member Functions

- Sound (char *path, int loop)

  Sound constructor.
- ~Sound ()

  Sound destructor.
- void play ()

  Plays the sound data contained in the class.

Private Attributes

- ALuint audioBuffer

  Binding id for sound data storage.
- ALuint audioSource

  Binding id for position, velocity, etc. of sound source.

3.9.1 Detailed Description

The Sound class loads and stores sound data.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Sound::Sound (char * *path*, int *loop*)

Sound constructor.

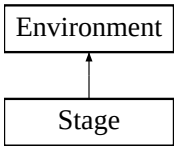
Parameters

path	Path to the sound file that should be loaded.
loop	Non-zero if the sound should be played as an infinite loop.

3.10 Stage Class Reference

`#include <Environment.h>`

Inheritance diagram for Stage:



Public Member Functions

- Stage ()
  Stage constructor.
- ~Stage ()
  Stage destructor.
- void addBoundary (cpVect p1, cpVect p2, ObjGPUData *gpuData)
  Adds a box boundary to the stage.
- void addDynamicObject (glm::vec2 pos, ObjGPUData *gpuData)
  Adds a dynamic object to the stage.
- void updateEnvironment (double dt)
  Function for updating the environment.
- void drawEnvironment ()
  Function for drawing the environment.
- void processKB (int key, int scancode, int action, int mods)
  Function for keyboard input processing.
- void processContinuousInput ()
  Function for continuous input processing.
- void processMousePosition (float xpos, float ypos)

Function for mouse position change processing.

- void processMouseClick (int button, int action, int mods, float winX, float winY)

  Function for mouse input processing.

- bool checkCompletion ()

  Function checks if stage is complete.

Private Member Functions

- void drawObj (Obj currentObj, bool isBoundary=false)

  Draws an object.

Private Attributes

- cpSpace * envSpace

  Pointer to the chipmunk space associated with the stage.

- std::vector< StaticObject * > boundaries

  Stored boundary objects (these are a special subset as they are drawn using different shaders than generic objects).

- std::vector< StaticObject * > staticObjects

  Stored static objects.

- std::vector< DynamicObject * > dynamicObjects

  Stored dynamic objects.

- std::vector< KinematicObject * > kinematicObjects

  Stored kinematic objects.

- float stageTime

  Time elapsed since beginning the stage.

- DynamicObject * userControlObject

  Pointer to the dynamic object that is controlled by the user (normally the hero object).

Additional Inherited Members

3.10.1 Detailed Description

The Stage class is derived from the Environment class and holds and handles changes to the game state when the user is playing a stage.

3.10.2 Member Function Documentation

3.10.2.1 void Stage::addBoundary (cpVect *p1*, cpVect *p2*, ObjGPUData * *gpuData*)

Adds a box boundary to the stage.

Parameters

p1	Lower left coordinate of boundary
p2	Upper right coordinate of boundary
gpuData	Pointer to the ObjGPUData that contains the gpu data used for the boundary.

3.10.2.2 void Stage::addDynamicObject (glm::vec2 *pos*, ObjGPUData * *gpuData*)

Adds a dynamic object to the stage.

Parameters

pos	Position of the centroid of the object
gpuData	Pointer to the ObjGPUData that contains the gpu data used for the object.

3.10.2.3 bool Stage::checkCompletion ( )

Function checks if stage is complete.

Returns

True if stage is complete, else false.

3.10.2.4 void Stage::drawObj (Obj *currentObj*, bool *isBoundary* = `false`) `[private]`

Draws an object.

Parameters

currentObj	Object that should be drawn.
isBoundary	Flag for boundaries (default = false); boundaries are drawn slightly differently.

3.10.2.5 void Stage::processKB (int *key*, int *scancode*, int *action*, int *mods*) `[virtual]`

Function for keyboard input processing.

Parameters

key	Key to which the action corresponds.
scancode	System specific key code.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)

Implements Environment.

3.10.2.6 void Stage::processMouseClick (int *button*, int *action*, int *mods*, float *winX*, float *winY*) `[virtual]`

Function for mouse input processing.

Parameters

button	Mouse button to which the action corresponds.
action	The action (i.e. button up, down, held, etc.)
mods	Active modifiers (i.e. shift, control, etc.)
winX	Mouse cursor x-position
winY	Mouse cursor y-position

Implements Environment.

3.10.2.7 void Stage::processMousePosition (float *xpos*, float *ypos*) [virtual]

Function for mouse position change processing.

Parameters

xpos	Mouse cursor x-position
ypos	Mouse cursor y-position

Implements Environment.

3.10.2.8 void Stage::updateEnvironment (double *dt*) [virtual]

Function for updating the environment.

Parameters

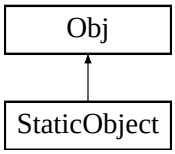
dt	Time step to be used for the update (time since last update) in milliseconds.

Implements Environment.

3.11 StaticObject Class Reference

`#include <Obj.h>`

Inheritance diagram for StaticObject:



Public Member Functions

- StaticObject (cpSpace *space, cpVect p1, cpVect p2, ObjGPUData *gpuData)
  StaticObject constructor.

Additional Inherited Members

3.11.1 Detailed Description

The StaticObject class is derived from the Obj class. Physics calculations are generally ignored for static objects. Used primarily for boundaries and stationary platforms.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 StaticObject::StaticObject (cpSpace * *space*, cpVect *p1*, cpVect *p2*, ObjGPUData * *gpuData*)

StaticObject constructor.

Parameters

space	Chipmunk 2D space to attach object to
p1	Bottom left coordinate of bounding box
p2	Upper right coordinate of bounding box
gpuData	Pointer to the gpu data associated with the object

Index