GLUB

PROJECT REPORT

GAME DEVELOPER

Angelika Bugl

Christoph Lipphart

GAME DESIGN

Philipp Gratzer

FH Hagenberg, IM

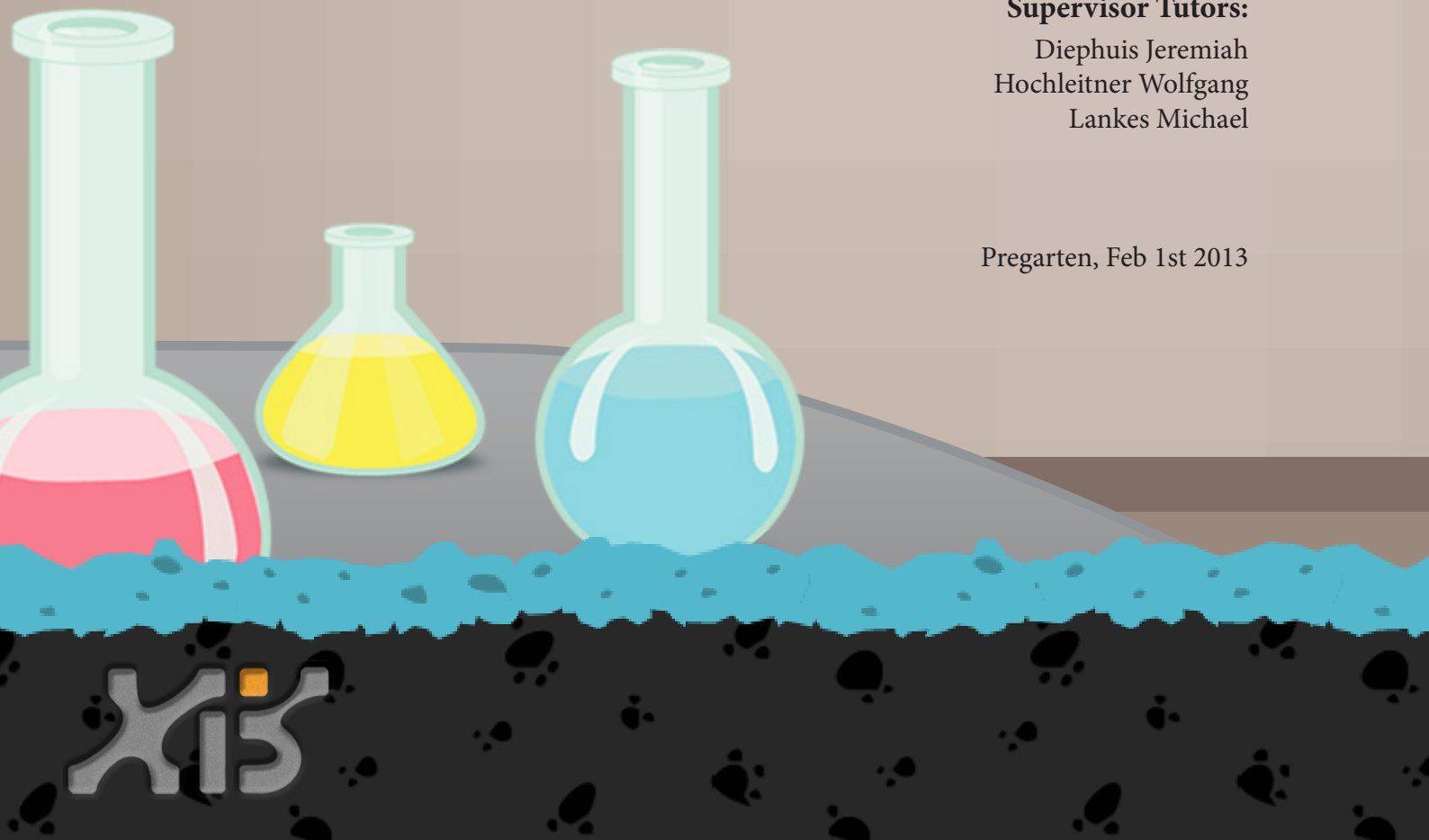
Cross-Platform-Game: GLUB
Semester Project 1
Winter semester 2012 / 2013

Game Developer
Angelika Bugl (IM)
Christoph Lipphart (IM)

Game Design
Philipp Grazer (DA)

Supervisor Tutors:
Diephuis Jeremiah
Hochleitner Wolfgang
Lankes Michael

Pregarten, Feb 1st 2013



GLUB

TECHNICAL SPECIFICATION

Side-Scroller Game

Platforms

Android, Mac OSX 10.7, Windows 7

Programming Language

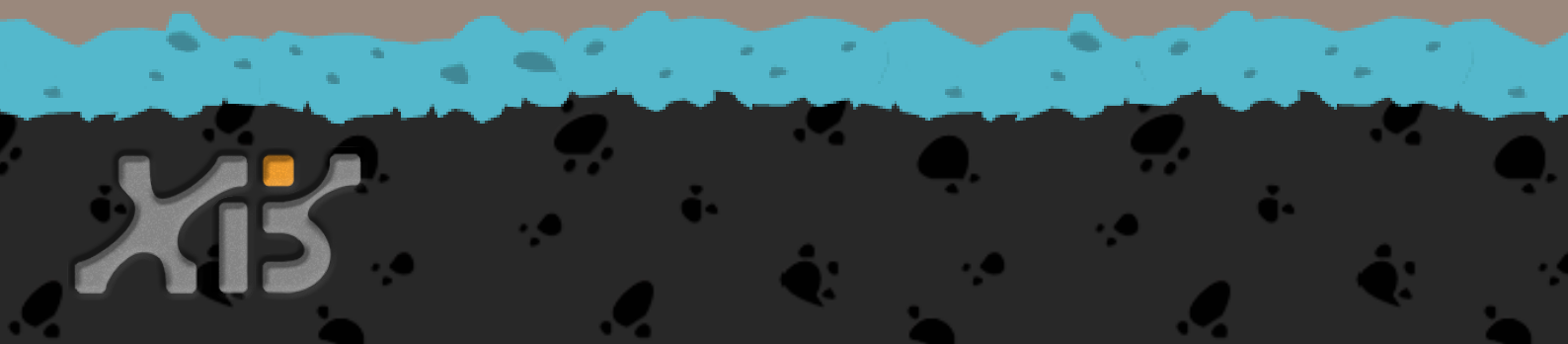
C++

Game Engine

XiS	self-made C++ engine developed as external static library. (xis-engine.com)

External Frameworks

OpenGL ES	When developing for mobile devices using OpenGL one has to use the light-weight version for embedded systems.
Rapid XML	To interpret XML in our Engine we decided to use Rapid XML, a light-weight and fast Open Source library.
Box2D	As we have already successfully used Box2D for the last semester project (MTD PRO5, Jaro) we decided to keep it as Physics Strategy in the xis-engine.
libPNG + zLib	To decode PNG files to draw it as textures in the OpenGL context we included libPNG which requires the additional library zLib (zip library).



GLUB

STORY

An old scientist experiments a lot and therefore has already produced lots of different mutant animals. Glub is one of them. The game starts after Glub ecloses out of his egg in a complex of test tubes. He is very curious and hungry and therefore he swims away to look for something to eat. But there are lots of risks waiting for Glub. The user will help Glub to survive in this dangerous environment where electric bubbles and barriers try to stop the unfamiliar creature.

GLUB

THE GAME

In the game the main character, called Glub, swims in a test tube. The player shall help Glub to survive all the risk, which will appear in the level. Therefore Glub can be navigated up and down using the mouse wheel on a computer or tilt the device when playing on a mobile device. In the test tube there are obstacles which have to be avoided and lots of different kinds of bubbles. Simple bubbles can be burst by a click or eaten by Glub. If Glub has eaten too many bubbles he will explode and the player has to restart from the last checkpoint. But bubbles will also be digested when the main character has not eaten any bubble for a longer period time.

Another kind of bubble is the electro bubble. It is necessary to avoid them, because the water will be energized when the character collides with it or the player touches such a bubble.

Sometimes there will be obstacles in a level, which can be destructed by throwing a smash bubble at them using the mouse or a touch gesture. Also simple bubbles can be destructed this way, but if an electro bubble was hit by a smash bubble Glub will also die.

The eaten bubbles have an influence on Glubs weight. Simple bubbles will make him light, smash bubbles will increase his weight. His weight influences how fast he can swim up or down.

Additionally there are also star bubbles in the level which make Glub thin again. Collecting them is necessary to unlock new levels.

GLUB

XiS ENGINE

We've worked on our Game Engine since the 3rd term of our Mediatechnology and –design studies at the FH Hagenberg. We began to write it in Java, then we've rewritten it into C++ for the mobile device platform bada in the 4th term of MTD and in the 5th term we changed it to use it on Windows platforms. Now we've developed a completely new concept, where it is a cross-platform framework and therefore our game can run on windows, mac and android platforms.

This caused lots and lots of changes and refactorings. Approximately all of the code is new or at least refactored. We just kept some of the rough basic concepts.

service-based

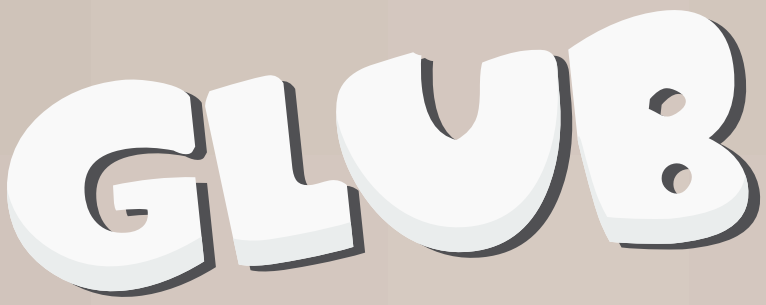
The xis-engine is a service-based Game Engine. The Services can be accessed using the singleton core. There are lots of services, which can, but not necessarily have to be installed. The developer can also create its own services and add them to the core, which will then update them, accordingly of the service type. This makes it easy to extend the Engine.

Test Suite

The xis-engine has now it's own Test Suite, which was used to test several services very intensive to avoid nasty bugs.

Input System

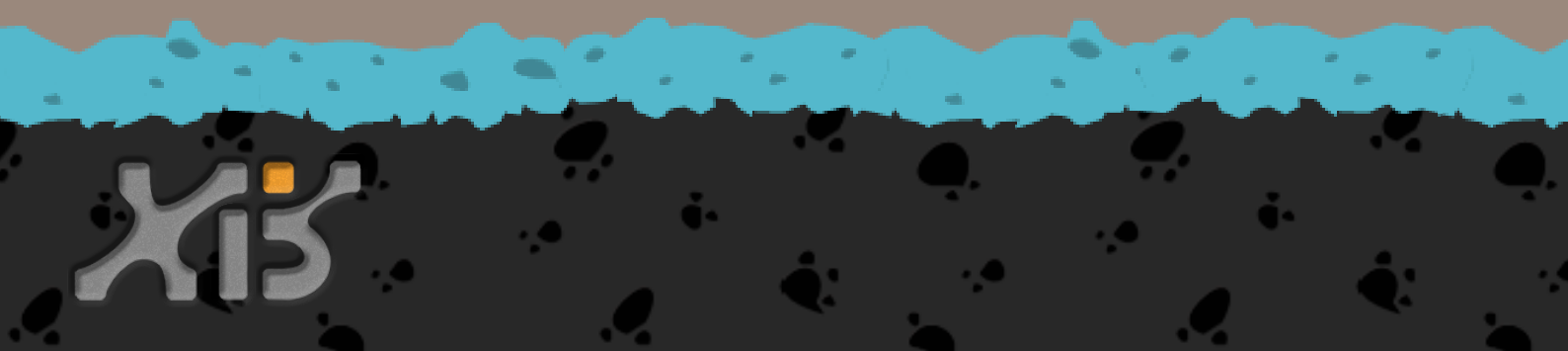
The InputSystem handles different input devices, including mouse, touch and keyboard. It also enables a developer to use the accelerometer of android devices as input.



XiS ENGiNE

Most important integrated services

EventSystem	is one of the most important services as it handles most of the communication between different services.
LoggingSystem	was refactored a lot and makes it easy to find bugs as all the logs can be written in an external HTML file which has filtering functions, like a search bar, filtering by log type (debug, info, error …) and supports information blocks.
EntityManager	has now a very new concept. It provides an EntityFactory which is responsible for the creation and destruction of objects. But if one orders an object from the factory this does not necessarily mean that it will create a new entity, but it may reuse an older entity which was intended to be destroyed before. This means that a pool of objects will be created, which can increase performance by avoiding lots of "new" and "delete" statements.
TimerSystem	which enables the developer to create several different timers and control them independently.
PhysicsManager	makes it possible to use Box2D using an uniform interface. It would also be easily possible to integrate other physics engines, for example if the game requires 3D physics, and use it by just switching the physics strategy.
GuiSystem	makes it easy to create a menu and place several clickable or non-clickable components in different sizes. The structure of a component is similar to the box model used in CSS having for example a margin, padding, width, height, left, bottom, …
Animation	It is easy to define new AnimationProperties and therefore it is possible to create animations for everything you can imagine using different kinds of easingcurves. The possibility to group them and make series of animations may also be very useful.



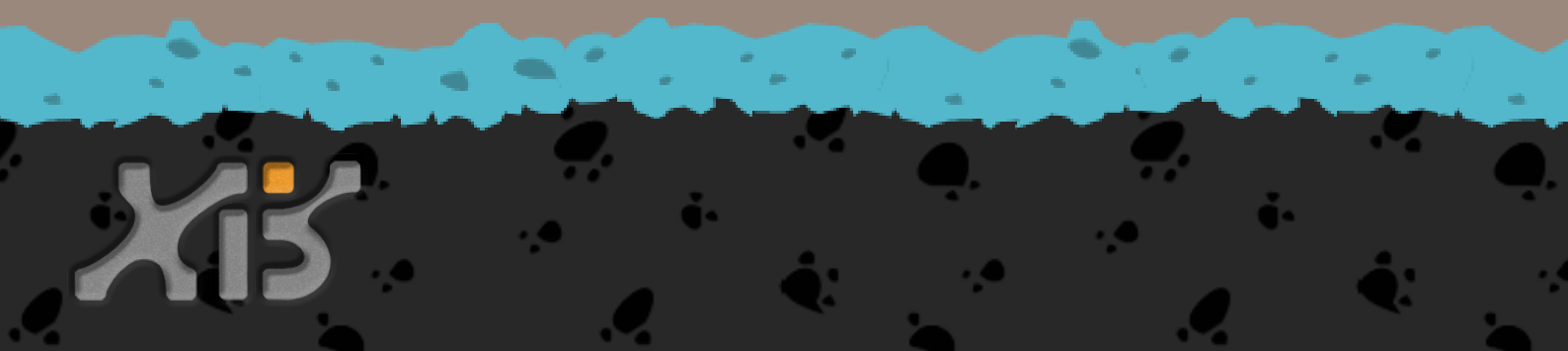
ViewSystem

The complex view system of the xis-engine makes it easy to create different views, which can have many layers. Every layer may have a camera attached. This makes it for example possible to show different extracts of a scene at the same time, create different scenes, switch between them in a comfortable way and create complex camera movements and zooms.

FileSystem

It is also possible to add shaders from external files into a project

It is easily possible to load data from external files. Therefore the developer can choose between two kinds of file formats: xml and so-called model files which are perfectly to quickly save some variables, regardless of which data type.



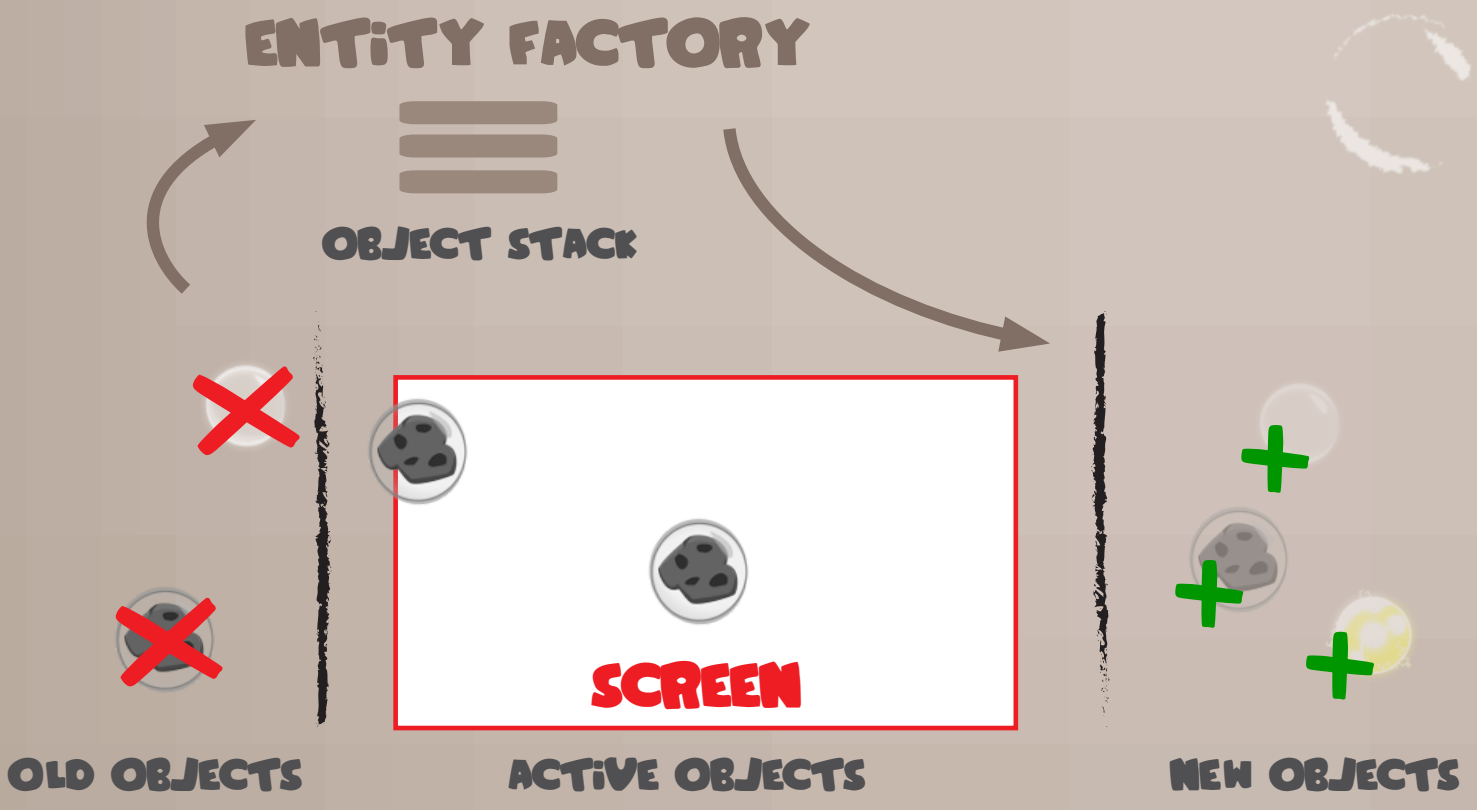
GLUB

FOCUS

DynamicLevelLoader

As this game is a sidescroller game for mobile devices we could not just load the whole level at the beginning. We had to find an algorithm which enables us to dynamically load and unload objects.

We fixed this problem by creating the DynamicLevelLoader. This class gets an ordered list of LevelObjectDescriptions as input. This list represents the whole level. The loader iterates over this list through the level and instructs the ObjectCreator to create or destruct the needed or unused objects when necessary. Therefore the class WalkThrough provides the current position within the level and throws a WalkThroughChangedEvent to notify everyone interested.



GLUB

FOCUS

Android

As our whole game is written in C++ we searched for a solution where we do not have to create a lot of Java code for the Android version. We found out that Android provides a native Java class Activity which calls a predefined main function from a C++ library. So we just had to write a particular main function in C++ and add a statement in the Android manifest which says that the native Activity shall be used with our library.

File input

It was very important that lots of things can be changed easily and quickly for making debugging more comfortable. We solved this problem by creating some config files to trigger the game and it's components.

Of course also every level is saved in an external file and will be read-in when needed. Therefore we created a light-weight xml-structure with which it is easy to create all the objects in the level.

We also saved configurations about the sprite animations of the character and the bubbles and the complete information about the GUI structure and images in external files.

Checkpoints

Our game also includes the concept of checkpoints. Those are represented by a big bottle which will appear at the front layer. For a better look we integrated a self-made ray-tracing glass bottle shader.

GLUB

FOCUS

Input

For input we used several different sources. On the PC or Mac we used some functionalities of the mouse: The wheel to navigate the character and the press, release and move of the left mouse button to destroy of throw bubbles. On android devices we decided to use the acceleration values to control the character and touch clicks and swipes for the interaction with the bubbles.

GLUB

FOCUS

Terrain

To make it easy to create the terrain, we developed an algorithm which automatically detects the corners and draws the blue decorations at the outlines. This is done by the TerrainLogic. This module gets notified by the DynamicLevelLoader every time when a new vertical line of terrain objects is loaded. It compares the last loaded object line with the new one. It iterates over the line from bottom to top and checks where to place a decoration. When there is for example a quadratic terrain object on the left side, but not on the right it knows that it has to draw a decoration there. When two edges lie on top there won't be such a graphic.

GLUB

FOCUS

Animations and Specials

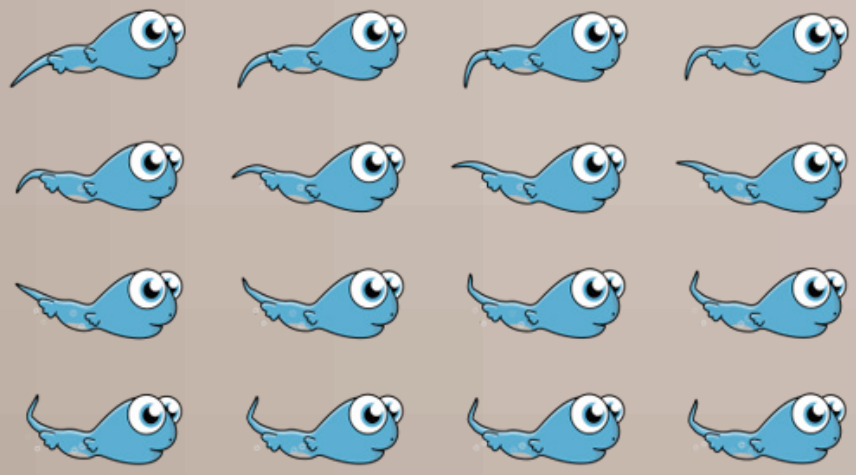
It was very important to us to develop a lively game. We found lots of possibilities to do that and could achieve to implement some of them:

We created PropertyAnimations in the GUI, for example we animated the position and the scale factor of the camera in the main menu. To create the illusion of a 3D perspective we animated the different images at the same time variable.

In the game we added some additional effects via sprite animations. The main character can for example blink.

There are also lots of effects in the game which are (nearly) only created by programming. The player blinks when it would die when it would eat one more bubble, it glows when it has eaten a star bubble, when it gets electrified by a shock bubble the whole screen flashes and when it swims upwards or downwards very quickly, it will rotate a little bit in this direction. Moreover there are special animations when a terrain or a bubble is destroyed.

We also added self-written shaders, like the background blur and deformation and the magnifying effect on the bottle, which represents a checkpoint.



GLUB

PROBLEMS

At the beginning it was very difficult to get everything running on different platforms.

First we attempted to use Eclipse as our development environment. It was very difficult to include all the different libraries and set up all the pre-processor variables correctly, as we did not know where to do it, what has to be done exactly and what we have to consider.

OpenGL

When we started to include the OpenGL functionality we found out that we can't continue developing using Eclipse. On Windows it wasn't possible to build the glew library correctly with the G++ compiler. We found out that we have to use the Visual Studio Compiler to link the OpenGL functions.

On Mac we didn't use the glew library but the OpenGL view from the Objective-C Cocoa API, Apples native object-oriented application programming interface. It is not possible till now to compile Objective-C code in Eclipse.

Gesture input

It was also not that easy to develop a comfortable way to throw the smash bubble using gestures. At the end we just discarded our original algorithm to swipe a bubble and developed a new version, which worked much better. First we tried to find out if the gesture was a click or a swipe first. We did this using the time between the click and the release and the distance between the two positions. Then we told the InputManager, which knows about all the clickable bubbles in a level about this input. Now we immediately delegate the input to the InputManager. This checks if a press event hits a smash bubble and remembers this bubble. If it gets a release event it checks if there was previously a smash bubble hit by a click and tells this object to swipe to the new position.

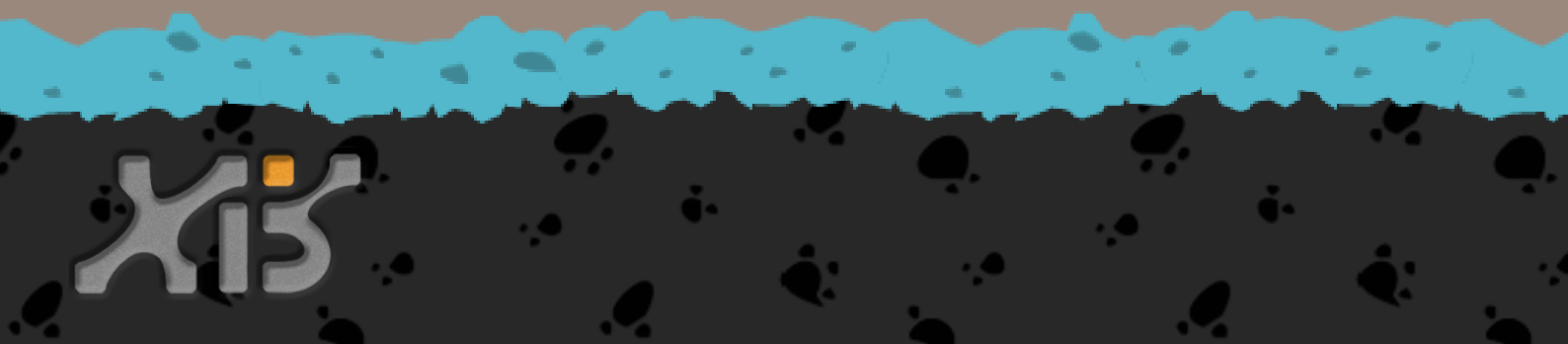
By adding a MouseMove or TouchMove listener we could also implement a graphic consisting out of several arrows, which visualize in which direction and with which speed the bubble will fly. Moreover we also highlighted the selected bubble to improve the usability.

GLUB

PROBLEMS

Time

But our biggest problem was time. It took us very long to develop a game concept. This was hindering us from working on the code for some time. We began working on the engine not till November. Working on the real game was not possible till the Christmas break. But after that we had a lot of time and motivation and we could quickly develop the majority of the features we wanted to develop, without having lots of nasty bugs. Very important therefore was the TestSuite of the xis-engine and plenty of Unittests we created. The easy use of the xis-engine also helped us a lot.



GLUB

FUTURE PROSPECTS

The game is not finished right now and we've planned to invest some more time into it to finish and extend it. There are some topics we would like to work on:

Sound	Right now there's no sound in the game as we had not have enough time to integrate OpenAL into our engine.
LevelEditor	We need to create more levels and therefore it would be useful to develop a level editor to make the creation more efficient.

To make the new levels more challenging and to keep them interesting it is necessary to design some new sorts of objects and bubbles.

Terrain	Gluey slime could be placed somewhere in the level. The character may get stuck when it touches it.
	Dangerous stings kill Glub.
More Bubbles	The poison bubble has not to be eaten by Glub otherwise it will die except he eats a star bubble, which cures him.
	The slime bubble encloses the character and makes him dead slow.
Opponent	We've planned to create a bubble creating mutant as an opponent at the end of every third level.
Particle System	We wanted to add a little particle system to the engine to add special effects and hence make it easy to polish the graphical representation of the game.

GLUB

USER DOCUMENTATION

SideScroller Game for Windows, Mac and Android

Goal

The goal is to navigate the character Glub safely through the level.

Navigation

Glub can be navigated up and down using the mouse wheel on PC and Mac or tilt the device on mobile phones.

You can click on bubbles and swipe smash bubbles by swiping them with the mouse or your finger.

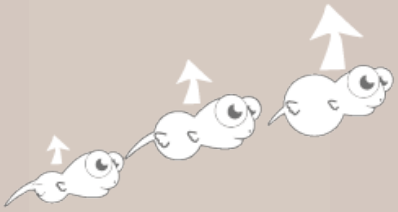
Explanations

Simple bubbles	You can click at them to destroy them. Glub can eat them, but if Glub eats more than 5 bubbles it will explode. If Glub does not eat a bubble for several seconds it will digest a bubble and get thin again. Glub will get more light-weight when he eats simple bubbles.
Smash bubbles	You can destroy other bubbles and Destructibles by throwing a smash bubble with your finger or the mouse onto them. When Glub eats them he will gain weight.
Star bubbles	If Glub eats one of them it will get thin again immediately.
Electric bubbles	Don't touch them neither with Glub nor with your finger or the mouse. Don't throw a smash bubble on them. Those bubbles are very dangerous because they can electrify the water which kills Glub.
Destructible	An obstacle which can be destructed by throwing a smash bubble at it.

AVOID EATING NORMAL BUBBLES...

... BECAUSE GLUB'S BODY WILL BLOAT...

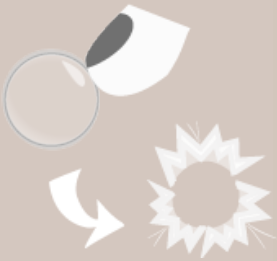
... AND THIS INCREASES BUOYANCY, MAKING NAVIGATION MORE DIFFICULT ...



AVOID EATING OR TOUCHING SHOCK BUBBLES

... AND AFTER FIVE BUBBLES, GLUB WILL BURST!

TOUCH THE DISPLAY TO POP THE NORMAL BUBBLES



EATING SMASH BUBBLES WILL MAKE GLUB VERY HEAVY ...

THROW THEM INSTEAD TO SMASH OBSTACLES

COLLECT STAR BUBBLES

THEY'LL MAKE GLUB THIN AGAIN