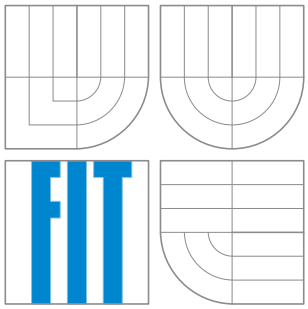
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY



BubbleShoooooter

Computer Graphics project

Ondřej Lengál, xlenga00@stud.fit.vutbr.cz
Libor Polčák, xpolca03@stud.fit.vutbr.cz
Boris Procházka, xproch63@stud.fit.vutbr.cz
Petr Zemek, xzemek02@stud.fit.vutbr.cz

2008

Contents

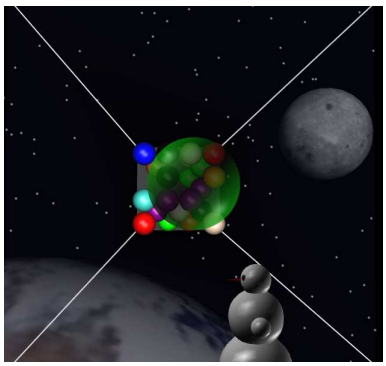
1 Introduction

This is a project documentation for BubbleShoooooter, a Computer Graphics project. The task was to create a computer game using the OpenInventor library. Our solution is a 3D variation of a popular GNU/Linux game called Frozen-Bubble (http://www.frozen-bubble.org). The principles of the game are following: the player is in front of a structure of static bubbles with various appearance; the task of the player is to use her bubble cannon to destroy the aforesaid structure. This can be done using one property of the bubbles — when there is a chain of given number of bubbles, all bubbles in the chain disappear.

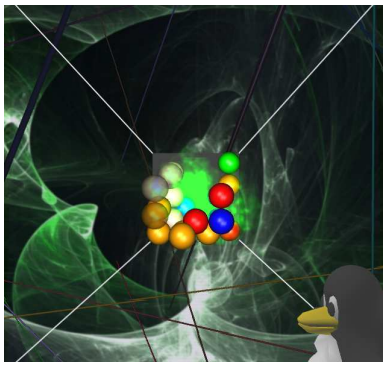
2 Project presentation

Each game is situated in some environment containing a play box, in which there is an avatar who launches bubbles towards the rear wall of the play box. The wall is moving towards the avatar in time. The goal is to remove all bubbles from the play box (before the wall hits the avatar) by shooting them on each other. When there are more than two bubbles of the same colour connected, the whole bubble chain disappears. Also, when some bubble happens to be not connected to other bubbles, it falls down and vanishes. User can choose different environments, play boxes and avatars (see example game screenshots).

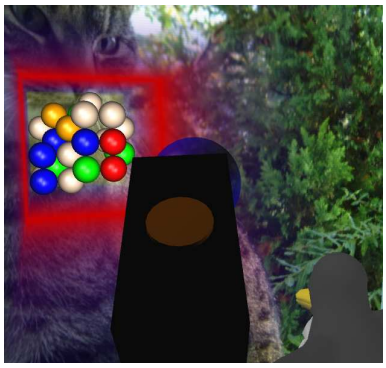
The first game screenshot gives an overall game overview and shows the default environment ("Space") with "Snowman" avatar.



The second game screenshot shows a different environment ("Wire"), different avatar ("Tux") and an explosion of a chain of green bubbles that were connected together.



The last screenshot shows yet another environment ("Album"), special "tunnel-like" play box and the bubble launching hammer in detail.



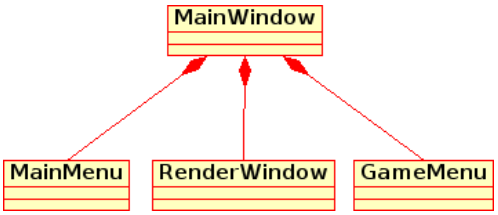
3 Game controls

Our game is really easy to control. Aiming (camera rotation) is done via mouse movement and to shoot a bubble, simply click the left mouse button. When you start a game, the mouse cursor is "captured" by the application, so you have to press the Escape key (`Esc`) to "release" the cursor.

4 Design and implementation

Basic implementation ideas and techniques are based on [1].

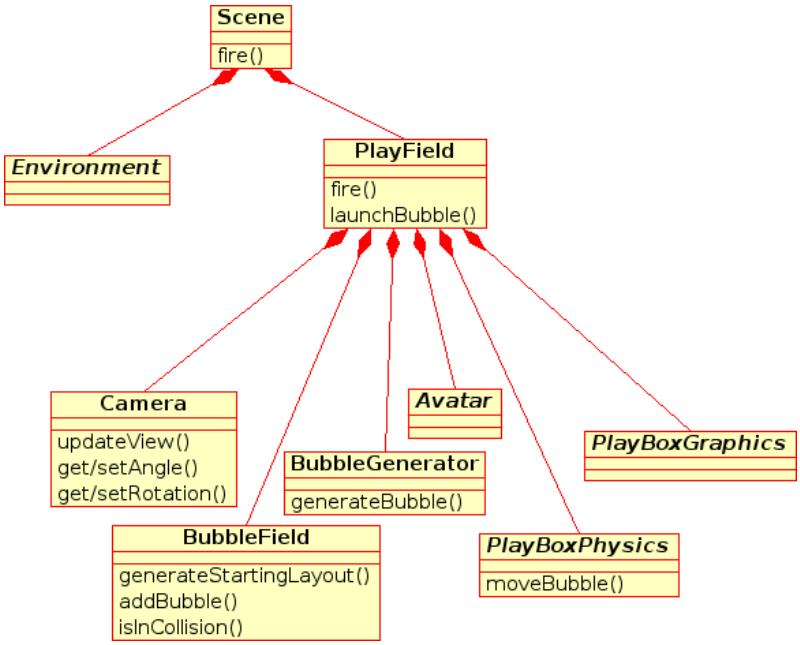
4.1 Graphical user interface



MainWindow represents the main window of the application. It is divided into two parts. The application menu and the playground. The application menu is implemented by *MainMenu*. The playground is used to show running game, new game options or additional game information. The *RenderWindow* builds the game scene. It creates camera, passes game options into *Scene* and registers callbacks that handle application events. The *GameMenu* creates new game menu, which is used to set game options (environment, avatar, difficulty etc.).

4.2 Game core

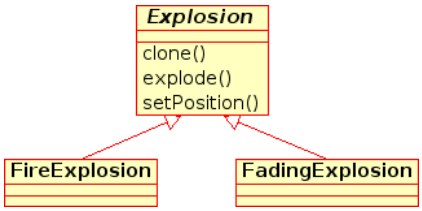
Scene represents the core of the game. It is designed to be a connection link between the GUI (main window, new game settings etc.) and the game itself. The *Scene* handles all communication



between the game core and the GUI. It resends user commands (such as firing a bubble) and requests for update in time in one way and ends the game in the other way.

Scene stores pointers to two very important objects. The first one is *Environment* (see subsection 4.3), which represents the graphical surroundings like stars, the Earth and the Moon in *SpaceEnvironment*. The second one is named *PlayField* and it takes care of the game itself.

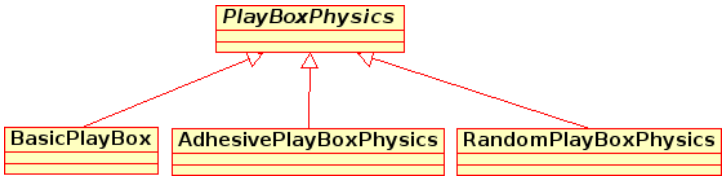
Main objects in the game are bubbles, which are instances of a class called *Bubble*. Bubbles are divided according to their type. Every type has assigned one colour and one type of explosion. Bubbles are able to change their position in the scene, change the direction of movement and explode when corresponding methods are called.



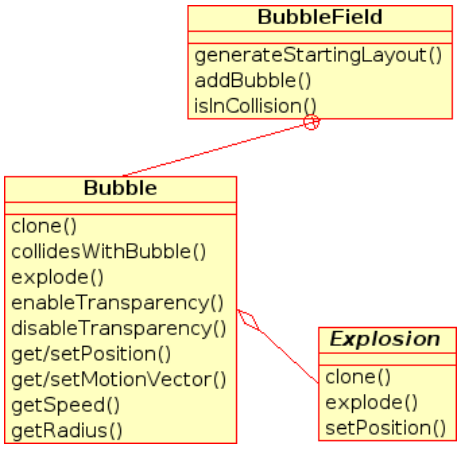
Each *Explosion* is able to clone itself (for copying reasons), set its position (the place for the explosion effect) and explode. The effect of the *FireExplosion* resembles a blast from a cannon and it is implemeted as a *sprite* (two-dimensional image integrated into the scene that is always turned directly to the *Camera*), while the effect of the *FadingExplosion* is a simple fadeout accomplished by modification of bubble transparency.

PlayField is responsible for the coordination of all objects that create the game experience. It stores pointers to *PlayBoxGraphics*, *PlayBoxPhysics*, *BubbleField* and *BubbleGenerator*. *PlayField* calls their methods when necessary.

PlayBoxGraphics (see subsection 4.3) is responsible for generation of a graphical representation of the *PlayField* (i.e. the walls of the playing field), while *PlayBoxPhysics* controls the flying bubble and properly adjusts its position and direction after it hits one of the walls. This implementation provides



an option to change the look of the *PlayField* and its physical behaviour independently.



BubbleField stores all bubbles that have hit the rear wall or another bubble that was added to the *BubbleField* earlier. The bubbles in the *BubbleField* are divided into plates of two different sizes — odd plates are a matrices of $n \cdot n$ positions and even plates are matrices of $(n-1) \cdot (n-1)$ positions (starting with a plate number 1). In this way, the bubbles can be stacked on top of another bubbles in the same way that eggs can be stacked in an egg carton. The first plate is connected to the rear wall, the second one is stacked on the first one towards the player avatar and so on.

When a new bubble is about to be added to the *BubbleField*, the nearest empty position to the position where the bubble has collided is chosen. The bubble is moved to its new position and it is checked if the just added bubble has been connected to more bubbles of the same type. In case this happens, all bubbles in the chain are ordered to explode and all bubbles that become unconnected with the rear wall are removed from the game.

Every bubble is generated in the *BubbleGenerator*, which creates an initial set of bubbles in the beginning of the game. The number of bubbles' types can be changed in the "New game" menu. The type of every bubble, which can be seen in the *PlayField* itself or in the *BubbleField*, is chosen randomly and the original bubble of that type is then cloned and the clone is added to the game.

In the beginning of the game, the *PlayField* is responsible for filling the *BubbleField* with bubbles according to the parameters which has been set in the "New game" menu. After the *BubbleField* is initialised, bubbles are added only when the player fires one. The motion vector of the fired bubble is set according to the position of the camera.

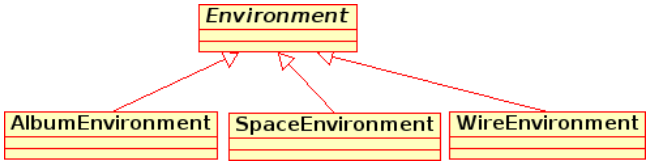
The player is in the game represented by their avatar (see subsection 4.3), which changes its behaviour according to the movement of the camera (for this purpose, camera implements the observer design pattern [2]). When the fire button is pressed, the avatar hits the waiting bubble and it begins its journey towards the rear of the *BubbleField*.

When one of the conditions which ends the game is fulfilled, the *PlayField* detects it and ends the game. The game is ended successfully when all bubbles have been removed from the *BubbleField*.

However, if the starting position for the bubbles is in collision with a bubble in the *BubbleField*, the game is lost.

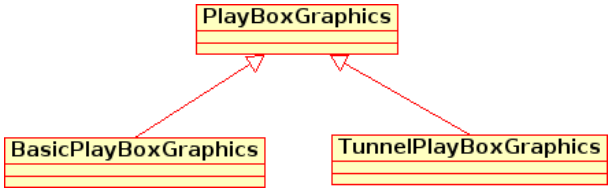
4.3 Graphical environment

In order to support modular design, the graphical representation of the play field is divided into two parts — the *Environment* and the *PlayBox*. Due to the fact that these objects generate the major percentage of graphics displayed on user screen, it affects the user's experience in a crucial way. Therefore, it has been given considerable attention.



Environment is a graphical background that causes no interaction with the player. Currently, there are three environments present:

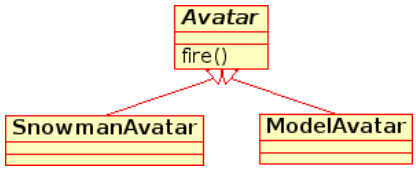
- **Space** — this environment tries to evoke the feeling of the player being situated in outer space. The background is created by a *skybox* (that is a box surrounding the player) with a star texture. This texture is generated on-line using a random number generator. A more immersive experience is achieved by dynamic objects of the Earth and the Moon. Both of these objects revolve, so the user experience is much better.
- **Wire** — a very dynamic environment with moving patterns.
- **Album** — this is a generic environment that creates a skybox where the wall textures are loaded from user modifiable album, therefore the user is able to substitute the default images with her own.



Play box is a graphical representation of the playing field, where the game takes place, so that the user can see the source of the object interaction in the scene. Currently, two play boxes are present:

- **Basic** — A basic semi-transparent play box.
- **Tunnel** — A play box with a visual effect resembling a trip through a tunnel. The feeling of movement is achieved by a movement of a texture with various coloured transitions and exploitation of correct texture mapping (repeating [3]). The texture is generated on-line as a 1-pixel-wide line of pixels that create a transition between two RGBA colours. The transition is a linear interpolation of all colour components (including the alpha component)

of both colours (i.e. red, green, blue and alpha) and the colours are easily substitutable with different ones. Movement of this texture on the surface of the tunnel then creates a feeling of movement through the tunnel.



Another very important graphical element of the scene is the *Avatar*, which is an object in the scene that represents the player of the game. Similarly to the abovementioned graphical elements of the environment, this one also uses modular design to enable good maintainability and modification. Two avatar types are currently present:

- **Snowman** — a complex model of a snowman. The snowman is built up from 3 snowballs (the body), 2 more snowballs (hands), 2 eyes, a mouth and a carrot (i.e. nose). An ice glitter effect was added to better simulate material properties of snow.
- **Model** — this is a generic avatar, that can have the form of any model loaded from an external file. Currently, a model of a penguin called Tux is used.

A part of the avatar is also so-called *Bubble Hammer*. This is the object that makes the movement of bubbles possible (as perceived by the player).

4.4 Open Inventor graph and changes in time

In order to handle the creation of the Open Inventor graph and to change it in time, two interfaces have been created.

Every class, which was designed to create objects with a graphical representation, is inherited from the *GraphicalNode*. Instances of these classes have their own *SoSeparator* or *SoGroup*, which are connected to the Open Inventor graph to their positions. *SoSeparator* is used when the object is transformed in some way, so other objects are not affected by operations which are done below the *SoSeparator*.

Classes, which can be updated in time because the instances of the class are moving or changing their states in some way, implement the *TimeUpdatable* interface, which has one virtual method that takes the time which is passed from the last call as a parameter.

5 Project management

Our team consists of four members. We identified the following main tasks of the application which had to be taken care of:

- graphical user interface, controls and connection with the main scene,
- functional aspect of the game,
- graphical aspect of the game, and

- individual application parts' interoperability.

Boris Procházka took care of the graphical user interface (GUI), camera, game controls and connection between the GUI and the rest of the application. *Libor Polčák*'s main area of interest was the functional aspect of the game, such as bubble shooting and motion, physics and correct play field behaviour ("glueing" bubbles together on collisions and bubbles removal). *Ondřej Lengál*'s specialization was the game graphics (in general), which includes the game environment, play box and the avatar. *Petr Zemek* was the team leader, who created the program skeleton and who was responsible for the overall functionality and cooperation of various application parts, i.e. the main scene and its components.

Because the overall project complexity required good design preparation, we had several team meetings where we were discussing the program design and where we were able to develop the application together. This was a huge advantage, because one could solve problems more quickly than using a nonverbal form of communication. For this purpose we used areas which our faculty provides for students.

Internet communication was also very important, because we live in different locations, so verbal communication (except team meetings) was possible only during the breaks between our lectures. For this purpose we used IRC and XMPP (Jabber) protocols and in order to manage our e-mail communication, we set up a mailing list.

We followed a specific coding stadard (e.g. code layout, naming conventions etc.) based on some widely used standards so that our source files would be easy to read and understand. As for the application design, we exploited some parts of UML to help us convey our ideas.

6 Conclusion

All specifications from the assignment were observed during the project development. Each member of our team was interested in the project and critical parts of the design were discussed together. We were focusing on playability (unplayable game is useless), overall graphical appearance (this was the main task) and clean, documented and readable source code (for easier modifications in the future), because these parts are very important in the result. Main parts were designed considering possible future development and extensibility, like adding new environments, play boxes, avatars, bubbles and physical behaviour of the play box.

The application was successfully tested on the faculty computers (GNU/Linux, x86) and on our systems (GNU/Linux, x86 and x86-64).

Used libraries

Standard C++98 library and STL
Coin 2.5.0 (http://www.coin3d.org/)
Simage 1.6.1 (http://www.coin3d.org/lib/simage/)
SoQt 1.4.1 (http://www.coin3d.org/lib/soqt)
Qt 3.3.8 (http://trolltech.com/)
Loki 0.1.6 (http://loki-lib.sourceforge.net/)

Used textures and models

Earth texture (http://www.root.cz/clanky/open-inventor-vesmirna-scena-4/)
Explosion textures (http://www.geocities.com/starlinesinc/)
Moon texture (http://www.geocities.com/cnlpepperplanet/)
Smoke texture (http://neverhood.etomite.sk/files/kourzelenej.jpeg)
Tux model (http://www.katorlegaz.com/3d_models/miscellaneous/0167/)

Metrices

Source files: 84
Source lines of code: 7703 (without empty lines)
Executable file size: 2.4 MB (GNU/Linux x86-64, g++ 4.3.2)

References

[1] Jan Pečiva. Seriál open inventor. [online], 2004. Available on URL: http://www.root.cz/serialy/open-inventor/.

[2] Kelpi Project. C++ observer implementation. [online]. Available on URL: http://kelpi.com/script/d73d5b.

[3] Josie Wernecke. *The Inventor Mentor: Programming Object Oriented 3D Graphics with Open Inventor, Release 2*. Addison-Wesley Professional, 1994.