Bender Haim 303740989
Gorelik Ilia 316946854

Computer Graphics, Exercise 3 : **3D Qbert Game**

Player Guide:

Game Instructions:

Lunch the application, resize to any window size you want, and start playing!
To view the controls in the game press **'/'** at any moment.

To start a new game press **'n'** from the start up screen. Beware that pressing **'n'** will start a new game at any moment, even in the middle of a level.

Controls:

Use **arrow keys** to move the qbert around. The Up arrow key will always move the qbert forward according to the **direction of his face**.

To change between the main view and the secondary view (small one in the right up corner) press **'v'.**

To Exit the game just close the application window.

In order to move the ariel view use your mouse.

Developer Guide:

Bonus features implemented:

- A first person shooter view.
- Multiple views (a smaller view at the corner of the screen). The name mechanism is used to put an additional view on the existing ones (a controls view for an example).
- Texture loading. (You can see an example on the toad object)

The structure and the design of the code:

The program has been developed in C++ using OOP design.

The whole program was separated to three main parts supporting the Model-View-Controller design pattern.

The game model which operates the whole game gets the input from the controller and passes it on to other models depending on the game state.
The Game Model is the one responsible for creating playing models (Called Qbert Models) each one of those implements a different level of the game.

The view part is separated to different views. The main view combines all the views together by putting them one on another. This design allows to add additional contexts on top of the image we already have (like score, lives, controls view and so on).

Logics behind the game:

The current level model decides when where to move game objects (qbert and enemies), when do they die, when do new enemies spawn etc.
The spawning of new enemies is implemented by using the factory design pattern (returning a new object from a string).

Each type of an enemy its own class implementing QbertEnemyObj.

Each new enemy (object) is created randomly on one of the boxes and is allowed to appear each period of time (first appearance delay is set separately).

Each time the qbert meets an enemy, a life is reduced and all the enemies are erased from the model.

Additional information:

The implementations uses the BOOST library (shared_ptr, random, format).
The graphic implementation uses SDL together with OpenGL
Source can be viewed using Visual Studio 2005.