Developing an Arcade Game

Results of a practical course at the Chair for Computer Graphics and Multimedia (RWTH Aachen University, Germany)

Sandra Hicks[*] Simon Hütz[†] Philipp Rainisch[‡]

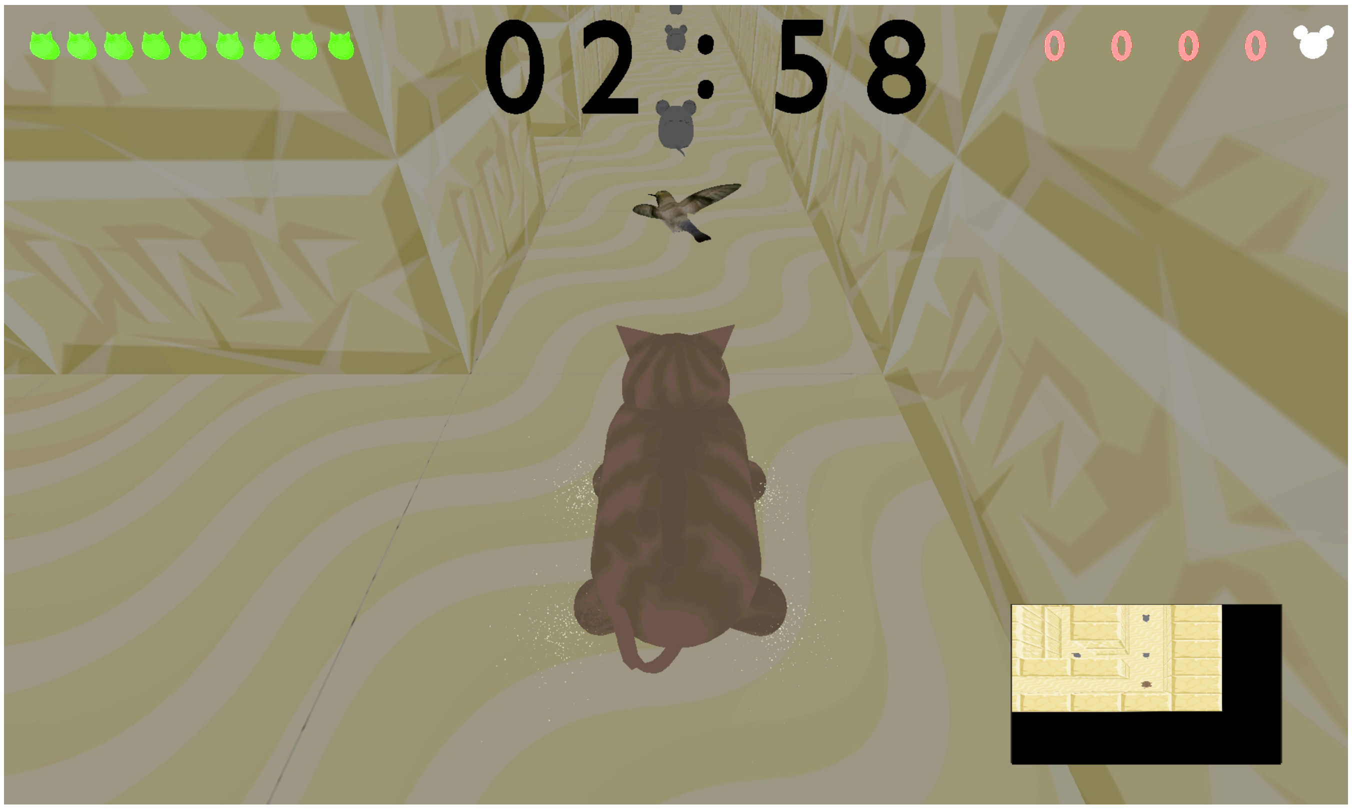

Figure 1: *Situation at the start of a new game in Catness.*

Abstract

This report is about the game 'Catness', which was developed in this year's practical course. The principle of the game is to play a hungry cat which tries to catch mice in a labyrinth. The goal is to catch as many mice as possible in a limited time. Life can be made either easier or harder by collecting powerups - this depends on your luck.

Keywords: game programming, arcade game

1 Game Concept

The main inspiration for our game is the old Windows^{TM} 98 Screensaver in which you run through a 3D maze. As our group mostly contained of cat fans, we decided that the protagonist must be a cat.

As you can see in Picture 1 one can see the cat with 9 lives which runs through the labyrinth to catch mice. The goal is to eat as many mice as possible in three minutes. Powerups disguised as birds can alter the gameplay in positive as well as negative ways. Among the powerups there is also a bomb, which can be used to destroy walls.

[*]sandra.hicks@rwth-aachen.de
[†]simon.huetz@rwth-aachen.de
[‡]philipp.rainisch@rwth-aachen.de

2 What we've done

2.1 Labyrinth

For our first implementation, we created a static labyrinth containing of walls and floors. Later, this labyrinth is generated randomly. At first our implemented algorithm builds a labyrinth like a star. At the beginning of the creation, it only has a starting point. From there all ways branch off. The decision in which direction the path goes on is taken randomly. After the creation of this star, the algorithm traverses the labyrinth and looks for dead ends. If one is found, the algorithm checks in which direction one of the walls can be destroyed to get to another path. This wall then is destroyed so we can guarantee that there are no dead ends. This algorithm can create labyrinths with an uneven side length. [Maz ; Dae ; Pri] As our labyrinth is generated randomly, our other gameplay elements must as well be spawned randomly. Among those are the torches, powerups and mice.

2.2 Collision

The cat is implemented with a collision box for the collision with walls, and mice and powerups are collected if they enter a certain radius around the cat.

2.3 Adding some intelligence

To make catching mice a little more a challenge, the mice have a certain intelligence. If the cat is far away, the mice will move randomly. But if the cat is near, the mice try to run away. This is achieved by implementing depth-first search. Up to a depth of 4 floor blocks, the maximum air-line distance between cat and mouse is chosen by the mouse to go on.

2.4 Powerups

To alter the gameplay powerups can be collected. These include positive ones as well as negative ones. Examples for positive powerups are 'fast cat' and 'freeze mice'. Negative powerups are 'catnip', which disturbs the vision of the player, 'baby cat' which shrinks the cat and stops the mice from running away. The cat can then be eaten by mice loosing lives. Beside the PowerUps we implemented bombs, which one can use to destroy walls and bomb your way free. The explode algorithm looks up which walls (up, down, left, right) can be destroyed. The range of the bomb is one field. If the cat is in this radius, the cat will loose a live. If a bomb destroys a wall with a torch, the torch will be respawned at another wall.

2.5 Content creation

To make the world come to life, we need models and textures for all the required objects. This includes the wall and floor blocks of the labyrinth, our actors, the cat, mouse and bird, the interface characters and symbols as well as the logo. The texture of the walls and floors are from [Sph] and the model and texture of the powerup bird are from [Hum]. All other models and textures were created by us in blender, exported as .obj and .jpg respectively and then imported and used in our game.

2.6 Change of scenes

In addition to the game itself, we implemented a 'Gameover' screen as well as a menu which shows the logo of the game and enables the user to start a new game or exit. If a new game is started, a totally new labyrinth is created.

3 Graphics

3.1 Lightning, shaders and shadows

In our game we employ different shaders. These range from simple ones to generate a depth texture for shadow maps to more complex ones which also compute a simple form of lighting and allow texturing. We only made use of vertex and fragment shaders, which also allows our game to run on older graphic cards. In our first implementation we used toon shading as our game should be in comic style, but a test with a basic implementation of phong shading gave better results and thus is used.
As the labyrinth contains torches to illuminate the scene, we also implemented shadows. At first we emulated the cube map with six simple shadow-maps. This resulted in good looking shadows which were quite costly on the computation part and thus did not allow more than four light sources. To allow more light sources, we

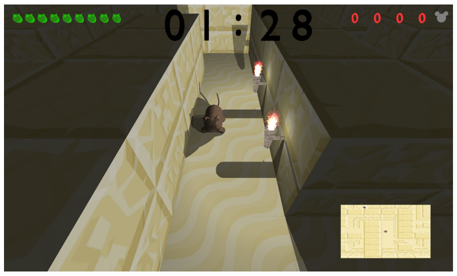

Figure 2: *This scene shows how shadows are casted from two light sources and the torch particle system.*

recalculate the shadow-maps every second frame. This basically allows twice as many light sources without a visible performance hit iff the game runs at around 60 frames per second. If the game runs slower however, one can see that the shadows have a certain delay. The second optimisation is a direct trade off between quality and speed. Instead of covering the shadows with six shadow-maps which cover 90 degrees each, we use three shadow maps which cover about 120 degrees ignoring the top as we do not have a ceiling. This again doubles the calculation speed, but makes shadows more inaccurate at large distances.

3.2 Lots of Particles

As another graphical effect we created a particle system which can be used for various purposes. The first application is a small dust cloud at the feet of the cat if it walks. The second is the flame of the torches and thirdly a big cloud of dust if a bomb explodes. The first attempt was to draw the particles by instancing, but as the number of particles was very limited and the drawing costly, we changed to using a single Vertexarrayobject per system for drawing all particles at once.

3.3 Interface

For a comfortable gameplay, we implemented an interface which shows the game's current state to the player. The interface shows the number of cat lives in the top left corner, the time which is left top middle and the score in the top right corner. If a bomb is collected, those are shown in the bottom left corner, and active powerups are listed left with an individual icon and a bar which shows how much active time is left. To make navigation in the labyrinth easier for the player, we also added a minimap to the interface. For this we render the fully lighted scene from above the cat into a frame buffer object, then use the resulting texture for a quadratic surface in the game.

References

Deadalus 2.5. `http://www.astrolog.org/labyrnth/daedalus.htm`.

Hummingbird. `http://www.3dvia.com/content/F5E94EEBFDCFE1F3`.

Maze generation algorithm. `http://en.wikipedia.org/wiki/Maze_generation_algorithm`.

Maze generation: Prim's algorithm. `http://weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm`.

Sphax texture pack. `http://bdcraft.net/`.