YALE UNIVERSITY

CPSC 478, final project description

Stanislaw Swidwinski, SS 2755

April 18, 2017

1 OVERVIEW.

For the final project, I have built a simple 3D version of the well-known game Pacman. The game is built using WebGL and may be played using the top-level game.html file. Please contact me at stanislaw.swidwinski@yale.edu if anything does not work!

The tree of files included in submission is described below:

```
|--drawables/
|   -- Contains javascript files describing the few simple geometric primitives
|      that are used within the game in different combinations.
|--drawing_sandbox/
|    -- Framework for easy creation of new geometric primitives. It is not
|       directly relevant to the game inself in the submission.
|--js/
|  |  -- Contains javascript files that define the game using the library files
|  |     defined within the lib/ directory.
|  |-- cameras.js
|  |  -- Defines the cameras and initialization of cameras
|  |-- drawing.js
|  |  -- Defines the way all primitives are drawn
|  |-- engine.js
|  |  -- Defines the life cycle of the game and puts together all of the pieces
|  |     to create the game.
|  |-- shaders.js
|     -- Defines the shaders used.
|--lib/
```

```
|  |-- pceng_config.js
|  |  -- The configuration file for the game. This is the file that specifies
|  |     the defails of the maps, the number of goats, the geometry of the map,
|  |     the geometry of waypoints used by goats, the number and positioning
|  |     of the dots etc.
|  |-- pceng_glue.js
|  |  -- The code that binds together the js/engine.js and lib/pceng.js files with
|  |     the html code within game.html.
|  |-- pceng.js
|  |  -- Contains all of the code necessary to represent objects within the game,
|  |     update their state, etc.
|  |-- spidergl-config.js
|  |-- spidergl.js
|--style.cc
|  -- Simple styling for game.html
|--game.html
|  -- The html file that allows us to play the game
```

2 How to play.

Use WASD to move and avoid the goats. The idea is exactly the same in the classic pacman game. Use the "2" and "1" keys to toggle through camera nodes.

Note that the game has goats instead of ghosts. This is by design – when talking about the game with a friend, she misheard "ghosts" for "goats" and it stayed in the game since.

3 More advanced aspects.

Within the game specification we were asked to specify and implement at least two more challenging aspects. I have chosen to implement collision detection, simple real-time random graph traversal using spline paths and animated models. I'll shortly discuss the implementation of each of the above and point to the files within the project where the actualy implementations may be found.

3.1 Collision Detection.

I have decided to implement efficient collision detection using spatial partitioning with uniform grid. This means that we create a grid whose elements "contain" drawables within the map that occupy any part of said element. Checking for a subset of drawables within the map that might collide with a dynamic object means just "collecting" the elements within given grids, which is constant time, since any dynamic object occupies at most four grids. The last assumption holds by design - the size of a grid is calculated so that the assumption holds.

Once the subset of elements that may collide is determined, the actual collision detection may take place. Within the game we only detect collisions between pairs of spheres, and a sphere and a parallelogram. Both of those collisions is simple to detect efficiently using analytical solutions for equations for said objects.

The implementation of the grid may be found within lib/pceng.js on around line 250. The object collisionMap implements the grid and allows us to query for possibly colliding objects. The functions within PCENG.Player and PCENG.NPC implement the actual geometrical checks for collisions. We use a priori collision detection, which means that we detect collisions before an illegal movement of an object occurs. The details of collision handlng may be found:

- PCENG.NPC._updatePlausibleCollisions
- PCENG.NPC._checkPackmanCollision
- PCENG.NPC._collisionWithWalls
- PCENG.NPC._updateState
- PCENG.Player._updatePlausibleCollisions
- PCENG.Player._collisionWithWalls
- PCENG.Player._collisionWithDots

3.2 Real-time random graph traversal on spline paths.

The goats within the game move across the map with almost no collisions, even though collision detection of goats with walls is not a part of the engine. This is achieved by passing to the goats a graph within the map that they may traverse using spline paths. The general algorithm for their movement can be described as:

Precomputation. This phase calculates the nodes and edges within the graph and assigns the nodes the corresponding coordinates. This happens within the lib/pceng_config.js file before any real-time part of the computation happens.

Initial conditions. Initial conditions are always fixed so that the goats begin at the point $(0,0,0)$ and begin moving towards the node of the graph the is directly above them.

Real time computation. Every goat internally holds four nodes and computes its path to the next node using a cardinal spline with the "s" factor of 0.15. That means that the goat holds in memory two "formerly visited nodes" and two "nodes to visit." Whenever a goat arrives at the next node, the oldest node is discarded and a new node is randomized, updating the spline.

Since splines are not arc length parametrized, we must moderate the speed of the goats. This is done by also computing the first derivative of the spline and using that function to estimate the proper step in the parameter for each goat. Details will be discussed below.

The code for precomputation may be find in detail in lib/pceng_config.js file. The actual

computation happens within ComputeMap.computeWaypoints(). For the sample map, any point at which there is a chance to change the direction of movement is a node and the edges are such that only legal movements may be made.

The code specifying initial conditios and real time computation may be found in lib/pceng.js file, within the PCENG.NPC object. To be precise:

- The *PCENG.NPC._updateWaypoints* function takes care of randomizing of new points to move to [1].
- The *PCENG.NPC._updateSpline* function takes care of creation of the spline using cardinal splines as mentioned before. Moreover, it computes the first derivative of the spline as well. Both of the functions are parametrized using a single parameter.
- The *PCENG.NPC._updateState* function makes use of said splines to update the position of the goats in time.

3.3 Simple animations

Simple animations of the objects within the game include the legs of the goats and the "mouth" of Pacman. The animation is achieved by introducing a simple clock within the js/engine.js file, which measures the time difference between consequent rendering phases and adjusts the positions of models. Since the models are hierarchical, this is very easily done.

To be precise about the implementation:

- The *PCENGClient.drawPacman* function within js/drawing.js shows how the mouth of the pacman is animated.
- The *PCENGClient.drawGoats* function within js/drawing.js shows how the legs of goats are animated.

4 Acknowledgements

The whole project relies heavily on the implementation of NVMC engine, which accompanies "Introduction to Computer Graphics, A Practical Learning Approach" by Fabio Ganovelli, Massimiliano Corsini, Sumanta Pattanaik and Marco Di Benedetto. The original code may be found at http://www.envymycarbook.com.

[1] I decided to disallow 180 degree turns in this case, but that is just to help with playability and disallow rather strange splines and sharp turns that result.