Distributed Systems – Final Project

Lukas Häfliger
ETH ID 11-916-376
haelukas@student.ethz.ch

Alexandra Maximova
ETH ID 09-913-534
amaximov@student.ethz.ch

Thomas Müller
ETH ID 11-946-936
muelltho@student.ethz.ch

Christian Vonrüti
ETH ID 11-930-914
cvonruet@student.ethz.ch

Alexander Viand
ETH ID 09-940-131
vianda@student.ethz.ch

Marko Živković
ETH ID 10-921-211
markoz@student.ethz.ch

ABSTRACT

We present a cross-platform game that allows up to eight players (who can be using different platforms)to play together via local network, or alternatively allows non-networked singleplayer against AI opponents. ■ The game is inspired by the "light cycle" scene from the 1982 film "Tron". ■ ■ The game is implemented using the Unity Engine, which is a high-level framework for game development ■ The game supports Windows, Linux and Mac OS on the x86 and x86_64 plattforms as well as iOS, Windows Phone 8 and Android. For this submission, we only feature the Windows x86/x86_64 and Android versions.

1. INTRODUCTION + PRIOR WORK

We were inspired to create this game after playing Armagetron Advanced, an open source game that is itself inspired by the light cycle scene from Tron.

Armagetron is one of two open source projects that try to recreate the light cycle scene, the other being GLTron by Andreas Umbach. GLTron is a more faithful reproduction both visually as well in terms of game mechanics of the original light cycle scene from the 1982 film while Armagetron Advanced is further removed from the original visually and offers a wide variety of gamemodes that go far beyond what is seen in the film. We were aiming more to recreate the enjoyment of playing Armagetron Advanced rather than to be faithful to the original scene. Since the scope of this project was very limited, we do not offer the extensive options and choice of game mode that Armagetron Advanced does, but rather focused on implementing the "core" game mode with fixed parameters. (Parameters being base speed, arena size, number of players, etc) ■ put param stuff in earlier, when desc. AA ■ Our game is visually very dissimilar to both the original scene as well as both games. Instead, we were inspired by the aesthetics from the 2010 film "Tron Legacy" which also features a light cycle scene, which does not, however, in terms of "game mechanics", bear much resemblance to the original. ■un-nest last sntc■

Armagetron Advanced offers local (split-screen), local network and internet multiplayer for up to ■????■ players. Armagetron Advanced itself is available for Windows, Linux and Mac OS but does not have an Android (or other mobile platform) port. There exists an android game called Androgetron developed by a member of the Armagetron Advanced community, however this is a much simplified clone which features only split-screen multiplayer. On the official Google Play Store ■TM!!■ there are a few more games ■list??■ that are essentially identical to Androgetron and seem to be all based on an android port of GLTron. They also do not offer any kind of non-split-screen multiplayer.

Split screen is a viable option on a full size laptop or desktop monitor, however it feels very cramped even on larger tablets and is essentially unusable on smartphones. Our goal was therefore to implement a true (networked) multiplayer experience on mobile devices. Instead of trying to adapt the large and complex code base of Armagetron Advanced (or GLTron), we decided to implement a version of the game with smaller scope from scratch. ■Mention OpenGL being waaay too complex?■ We decided to use Unity, a high level game engine and Integrated Development Engine (IDE) (which will be discussed in section 3), to implement the following game:

2. THE GAME (EXPLAINING)

We simplified AA to a single game mode, with fixed speed, arena size, etc. We did, however, also extend the game to new concepts not present in any of the other versions. Specifically, we introduced collectible powerups and moving obstacles that roam the arena.

1. you drive around in an arena

2. the bike moves constantly and you can steer only by turning left or right at 90° angles.

3. you leave a solid trail that is like a wall (and will be referred to as wall from now on)

4. colliding with your own or another player's wall results in your death

5. the aim is to be the last player alive

6. being close to another wall increases your speed, allowing you to e.g. overtake other players.

7. There are two different types of "power ups", which are randomly spawned items that will either increase your speed or make you invincible (you will be able to go through walls) for a short amount of time

8. there are (optional?) obstacles in the form of large cubes that slowly "roll" around the arena, collisions with these also result in death.

3. UNITY

Unity, which is developed by Unity Technologies, is a relatively new game engine that is designed "to make game dev more democratic" ■find cit■. Unity includes the core engine which is used through an IDE that offers an editor ■fig■ that allows manipulation of 3D environments, game assets and live debugging. Unity is based on Mono (an open source .NET-compatible framework) and allows development in JavaScript, Boo or C#. We chose to use C# because of its similarities to Java and Unity's integration with the extremely powerful Visual Studio IDE. Unity supports a large number of plattforms, including x86/x86_64 (Windows, Mac OS and Linux), Android, iOS and Windows Phone 8, with a generally very small required effort for porting between platforms. The support for native x86 versions greatly reduced the time needed for testing and debugging during development.

Unity's core concept is that of a "scene" which is an abstraction of a 3D space that contains "GameObjects". "GameObjects" are containers that contain "Components" (which can also be GameObjects, allowing for nesting). ■containers containing containers contained in ...■ All "GO"s have a basic "transform" "component" which contains information about position, rotation and velocity of the "GO". There are many other types of components, including "Lights", "Cameras" and "Materials" as well as Scripts.

Unity offers two very different methods for networking. A high-level "synchronization" feature as well as comparatively "low level" Remote Procedure Calls (RPC). Both require a GO to have a "NetworkView" component which makes them visible to the network in Unity.

Synchronization allows for high-frequency, low-cost updates of GOs. It can however only be used to synchronize certain properties of the GO, e.g. the transform (pos,rot,vel) component.

RPC on the other hand, is intended to be used for less frequent and less time-sensitive communication and is very similar to RPC implementations in other frameworks.

Our game combines (as ■ will ■ most Unity applications) both methods, using Synchronization for player and wall positions (or more correctly player and wall "transforms"), as well as RPC for state transitions and general control, as described in the next section.

Unity does not have a high-level concept representing the current application insth-level concept representing the current application instance in its entirety, instead every script has to be attacance in its entirety, instead every script has to be attached to a GameObject and each instance (client?) will execute the same code for a given GameObject (the code will usually have a currentObject.isMine() or Network.isServer type switch) It is therefore DESCRIBE HERE HOW IN-GAME SYNCH WORKS

While most of the networking during a round of the game is done via Unity's synchronization, the actual round (or game) start and end, server joining and other state transitions are done via RPC and event based programming.

4. STATE/CONTROL FLOW: NETWORK START GAME/NEXT ROUND

Their is an "empty" (i.e. code-only) GameObject that contains the "MainController" which is responsible for updating the GUI and communicating (locally) state changes between the Game and the NetworkController.

1. DiscoverServer

2. Lobby/Playerlist

3. StartGame

4. Death and new Round

5. COLLISONS/POWERUPS/PREDICTION

6. OPPONENTS: AI + CUBES

In local singleplayer, there wenn KI stirbt hat sie 9950/50 links/rechts.

Nur jedes 2te frame darf sie sich drehen (sonst geht es auf der stelle)

0.005

Sch... simple aber effektiv.

—

7. CONCLUSION

Give an overall conclusion that summarizes the main challenges you encountered and your lessons learned.

8. REFERENCES

[1] Services: Sending Notifications to the User. http://developer.android.com/guide/components/ services.html#Notifications. Accessed on 29 Aug 2013.

[2] E. Burnette. *Hello, Android: introducing Google's mobile development platform*. Pragmatic Bookshelf, 3 edition, 2010.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, 1999.

[4] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, UC Irvine, 2000.