Commercial Games Development Personal Diary

* 25/09/13
  + The specification was assigned.
  + Formed a team: “Gecko Faction Games”, with Anthony Lee, Alex Baden-Hinsley and Robert Cale. We brainstormed ideas on game mechanics, features and style to be pitched in the following session. Ideas included having different biomes to influence terrain and player physics, types of terrain features such as big jumps and lengthy descent sections, and a mechanic to make our game party-play orientated and unique: the King of the Hill system, where players would have to hold an area, that followed the group of players on the track, the longest to score points.
* 02/10/13
  + We pitched our game idea. Producers were unhappy with the King of the Hill gameplay element, but did suggest we find another way to implement an alternative method of scoring, along with traditional ‘first across the line wins’ racing mechanic. They liked our terrain ideas and suggested we focus our game on those elements.
* 09/10/13
  + We discussed what classes we would need and created a class diagram in our shared documentation. We all began work on the code base.
* 16/10/13
  + Some base code was imported into the SVN repository: GameObject, PhysicsObject and Camera classes are now in the project. These classes allow simple physics objects, with velocity vectors and orientation matrices, to be drawn onscreen.
* 23/10/13
  + We added and updated some base logic for GameObject, VBGameObjectBasic and IGameObject classes. These would allow us to create models and import them via text files, where a list of vertices and indices would be imported to the relevant lists to create a model. A simple Phong shader fx file was imported, with per vertex colouring to simply yet clearly show the objects onscreen.
  + Imported some basic models for debugging, including a simple car shape and icosahedron.
  + I began code for Track, including TrackSection and TrackManager. The track manager holds a list of TrackSections, where each track section would be connected to the previous one. This would take the length of the track piece and place the current piece according to the length of the last piece. The plan is to start with a simple straight line, but build ‘next placement’ position into the models, so more complex pieces can be imported to create an interesting yet coherent track.
* 30/10/13
  + Added classes for objects in the game world, such as base Car class and Powerup class. The car class holds all data relevant to a car object, but not player information, which will be delegated to a separate Player class that holds an instance of Car. Powerup is a base class, where each subclass is the specific power up that needs implementing. The base Power up class has virtual activate and deactivate functions, and an update that moves a power up’s position up and down like a sine wave.
  + I added more updates to base GameObject classes as necessary.
  + I also continued working on track generation, with more varied track sections and placements.
  + Simple vehicle model files were added to test model loading algorithms, with extra variables held per model file such as a model’s diameter and centre point.
* 06/11/13
  + I added third person camera controls for easier debugging. This allowed the user to move the camera around with the mouse and keyboard, to easily view models and check for errors in track model generation.
  + More model files were added, and I continued to clean up some unnecessary code in the Camera and Terrain classes.
  + I also added logic to allow the cars to move on key press. This was currently simplified, where checking the keyboard’s state for the keys as in the specification (QWE, ZXC, P[], M,.) would simply move the car forward, left or right. However, to keep the idea that turning a side rocket on would cause a centrifugal force, turning on the right side rocket would push the car left and vice versa.
* 13/11/13
  + I did lots of bug fixing and scrappy code cleaning in existing classes, particularly in Terrain due to algorithm testing.
  + I started the Player class implementation, with player count and player controls according to specification. The player class holds data such as their car instance and variables for their player keys, so each player would have a different car and boost keys. The class also has a handleControls method, to apply the correct forces to their car upon appropriate key press. This is also controlled by the global GameState, so pressing the boost key in the menu would randomise the car, while pressing boost during the race would give you a boost.
  + I also implemented a Pac-Man style loop around in the car class where going off-screen would lead you to loop around to the other side, adding a nostalgic feel for the retro-themed specification.
* 20/11/13
  + A configuration text file was added to allow for easy configuration of player count, resolution, controls and other variables.
  + I also fixed bugs in ChaseCamera, Car and Terrain classes. Fixed bugs in Car subclasses, as well as fixed a bug where a car and its collision model weren’t synched up correctly. This would happen because the collision model was set before a car’s position had been updated, so they would be out of sync.
* 27/11/13
  + I properly implemented a state machine for various game states. The game now has a 30 second countdown at the MENU, a 5 second rolling-road-style buffer in MENU\_TIMEOUT, and a proper PLAYING and RACE\_OVER state. This also fixes the bug where players could randomise their car for the whole race. This was applied to the Game1 class as well as the Player class.
  + I added the Flag class and began work on flag logic and scoring. Score is increased according to how long the player holds the flag, and colliding with the flag carrier will steal the flag from them. The flag’s parent player is set during collision checking, and being the flag’s parent passes the check to give an increase in score.
* 04/12/13
  + Updated physics and collision checking methods. I added a function, bounceObjects, to appropriately bounce two colliding physics objects away from each other according to their velocities and approximated centres of mass.
  + I also added base code for boost handling mechanics, such as building up the boost meter and using up a reservoir of boost, as well as ensuring players can only boost if at least half of their boost meter is full. Each player has a maximum boost amount, BOOST\_LIFE, of 100. If the current amount of boost, currentBoost, doesn’t equal BOOST\_Life, then increase the amount of current boost. For scoring points by holding the flag, a timer is used to check if it has been 1 second since the last score update, and increase score accordingly.
* 11/12/13
  + I fixed a bug where cars would fall through the track. Collisions were not being checked in the MENU\_TIMEOUT state and were instead being built in the playing class. This meant the cars would fall through the track in MENU\_TIMEOUT, but would collide if the state was PLAYING. This was simple to fix, by adding a collision check in MENU\_TIMEOUT.
  + We had our Alpha presentation – producers didn’t like our current terrain generation, so we discussed a method to redo it to their liking by generating a curve and building a vertex list from that. They also thought our vehicles were a bit cliché, so we would make more vehicle designs. They were happy with the retro/Tron aesthetic style.
* HOLIDAYS
* 22/01/14
  + Beta presentation. Producers happy with changes so far.
* 29/01/14
  + I added in a HUD class with needed logic to draw required HUD elements, such as the boost meter and player score. These elements are coloured per player according to the Player’s playerColour variable. I added some HUD sprites which I had created to the project.
* 05/02/14
  + Fixed some HUD code to position relative to the screen size as opposed to hard coded in. The original positions looked appropriate at the default resolution, but would be completely wrong and inappropriately scaled at other resolutions. This was fixed by placing each HUD element using the window’s Width and Height variables.
  + After, I continued working on the Powerups. I began work on building and checking collision models, for activating the power up instance.
* 12/02/13
  + I updated the PowerUp class to be appropriately scaled for the game world, coloured according to specification and to more believably bounce and hover in position.
  + I also discovered a bug with Sprite3D where the sprites were incorrectly drawn in reverse. To fix this, I had to change the loading order of the indices. The texture would be placed beginning at the ‘right’ side of the indices instead of the ‘left’, so they were swapped.
* 19/02/14
  + I updated the scale powerup to properly rescale players on contact, disappear, and to deactivate after a certain period, setting the original scale and thus returning the player to normal size.
  + I finished the building of collision spheres for powerups and completed logic to appropriately check for collisions and activate/deactivate powerup as necessary.
  + I also moved some boost logic to the Car class to make it easier to debug and follow correctly, as well as made the flag wave as if it was in the wind. The flag would adjust its Yaw by a sine wave, and add that to its current Yaw. This would allow the flag to turn with the parent car as well as wave semi-realistically.
  + I added logic to thrust and terrain powerups. The thrust powerup would simply give max boost to the parent player for a set amount of time, and the terrain powerup would call the biome change function in terrain manager.
  + I updated the Terrain classes. Terrain now holds ‘target’ colour values to allow for smooth colour changing rather than instant switching. This gives a visually pleasing change in colour when the terrain changes.
  + I also fixed a small bug with boosting where 2 players colliding for boost would accidentally get the same boost meter value (and hence unfairly receiving or losing boost) rather than individually updating their own values.
* 26/02/14
  + I began a 3D particle system, to allow for particle effects in 3D space. Initially, instantiation for the system was slow. The class would create a list of texture quads simultaneously to particles themselves. I needed to make a list of texture quads first and send one to a particle as a constructor parameter. With this in mind, I optimised instantiation.
  + I made an emitter base class, and made an explosion emitter from it. The base class would update each particle from a list of particles, as well as virtual functions for emitting particles. The explosion emitter would emit a particle with a given speed at a random direction from the emitter’s position.
  + I created some particle sprites and tested the emitter by emitting on car collisions.
  + I added a particle effect for side rockets when each side rocket is on.
* 05/03/14
  + I added more particle effects and sprites.
  + I gave the Powerup base class a simple particle effect to give them more visual flair.
  + I made the boost bar HUD elements dynamically change colour according to how much boost is left. The, I cleaned up some code in HUD to make it more efficient.
  + HUD now changes according to if a player is alive or dead. If the player is alive, the HUD will act as normal, but if they are dead, the player colour for the HUD will change to a dark grey. This helps visually inform the player that they are not in play and can’t use their boost.
  + I noticed a glitch in the Player class. Players would gain boost even if they aren’t in play. To fix this I simply added an IsAlive check to boost gaining. The boost style of a long change in velocity didn’t suit our game, so I changed boost to a shorter shunt-type movement. Players now shunt in the direction according to what buttons they are pressing – both side rockets = forward shunt, one = sideways, none = backwards.
  + As a TrackManager test, I changed TrackManager to now randomise between 3 separate biomes that change terrain colour and modify physics. There would be a simple terrain type with default settings, a bouncy terrain with a light blue colour scheme and bouncy physics, and a post-apocalyptic style terrain with a brown colour scheme and heavier physics.
* 12/03/13
  + I refactored Sprite3D class to make it more efficient and flexible, such as allowing for dynamic rescaling of texture quads.
  + I implemented more dynamic chase camera logic using spring physics, camera now chases the average position of the cars with a more natural motion.
  + I also added a function to allow changing the direction of the chase camera, for more dynamic motions and camera events. The direction would place the camera’s position accordingly, so setting the direction to, for example, Vector3.Left and multiplying this as necessary to get the camera an appropriate distance would cause the camera to chase the subject from the side on.
* 19/03/14
  + I began work on the Bomb, an item for players to avoid that works with a pass-the-parcel type mechanic. Like the flag, touching another player would pass the bomb to them. After a certain amount of time, the bomb would explode, killing the player and potentially launching nearby players.
* 26/03/14
  + Bomb bug fixes and refactoring.
* 02/04/14
  + Continued Bomb implementation. Bomb now has collision models, and players can pick up the bomb when they make contact with it. This parents that player to the bomb, and the bomb will float above the player to visually notify this.
* 09/04/14
  + I made the Bomb properly pass between players, parenting appropriately. Another visual notification for bomb was added: the bomb scales up and down according to a sine wave and increases fluctuations as time continues, and the bomb gets redder as the explosion time draws near.
  + In the HUD, I implemented stat bars In the MENU state, where each cars stats are displayed via rectangle bars according to each stat value. Currently static, would be more visually interesting if the bars moved to new values as cars are randomised rather than setting them immediately.
* 16/04/13
  + I made the HUD dynamically change stat bars. The bars now move towards new values for the car rather than immediately set, using a current length and comparing it to a desired length, where the desired length is the current stat value.
  + I created a new body model for car randomisation.
* 23/04/14
  + I implemented stats into the boosting calculations, so boosting now works according to the car’s shunt strength, weight, and how much boost they have left. The remaining boost bar functions like a pressure value, in that lower amounts of remaining boost mean you don’t boost as far and vice versa.
  + Adjusted all model stat values to compensate for new shunting method.
  + Made and added a new Main Booster model, and 2 new Wheel models.
* 30/04/14
  + I implemented a feature where Cars now hold who collided with them last, by that car’s integer Player ID, for a short duration of time. This allows for points to be given to the player that killed the other player, as well as determine if a death was caused by another player or if the death was a self-destruct, deducting points appropriately.
  + Added a ‘ShakeCamera’ function to ChaseCamera. Calling ShakeCamera with float magnitude and float duration causes the camera to randomly shake about its position and look at target for that magnitude and duration of time.
  + Implemented HUD system where the flag and bomb carriers have icons above their personal HUD. I also implemented a similar system that draws a notification of race pole positions.
  + Refactored newer HUD code to improve efficiency, and realigned text positioning.
  + I updated the Player class. Players can now veto the current track during the MENU state, where if all players veto a new track is generated. This is also reflected in a HUD notification.
  + I integrated the DPSF Particle library; cars now leave neon-style trails behind them.
  + Bug fixing, including a black screen bug where after all players had died, the camera didn’t have a look at target and was reset to null position, bomb class not reinitialising after game reset and other miscellaneous errors.

Mathu Watts

11022133