# Abstract

UGTA is a game designed to bring about mayhem and destruction. The user moves around the scene in the form of a car which has a cannon from where it fires bullets to take down buildings present around the scene. The game is played within a time limit. Upgrades are obtained on shooting down a particular number of building blocks, such as evolution of a car into a monster truck, shooting out harpoons, increase of time limit and more. The objective of UGTA is to shoot down as many buildings as possible so as to get your score as high as possible before time runs out.

# 1. Introduction

## 1.1 About the Game

The game, UGTA pays homage to classic single player 8 bit games of yesteryears. Though much more advanced than an 8 bit game, the look and feel resembles that of Minecraft. The gameplay is inspired from games like Grand Theft Auto and Crazy Taxi where for every point earned, the player gets an upgrade. The game starts with the player having controls of a car. The car comes with a canon fitted on top ready to shoot bullets. The player shoots at different buildings in the game to earn destruction points. After earning certain points, the car modifies in shape and engine capacity. There are two shape upgrades on the car. First one is a glider mode where the car glides over the ground with reduced friction and increased speed. The second one is monster truck mode where the car becomes more bulky and have higher engine capacity. Also there are upgrades for type of bullets that can be shot (harpoons) and total time of gameplay.

The main USP of the game is simulation of real world interactions in game world. The realistic effects of building blocks falling or the car colliding with a building are made possible with the use of Bullet Physics library. With help of Bullet Physics each object in the game world can be given real world properties like mass, gravity coefficient of friction etc. The physics behind the interactions is simulated with help of Bullet APIs. The gameplay experience is enhanced by background music. Also integrated in the gameplay are sounds of bullets being shot and engine revving

## 1.2 Bullet Physics

Bullet Physics is a Real-Time Physics simulation Library that can simulate collision detection, soft and rigid body dynamics. It has been used in several video games as well as in movies. Grand Theft Auto V, Megamind, Sherlock Holmes are some famous examples that use Bullet physics.

Bullet physics library is free and open-source software. It provides us with a rich set of features that made it a primary choice for development of our Game. It also hosts a detailed user manual, API Documentation and several tutorials to get started with.

Some of the features are:

* Rigid body and soft body simulation with discrete and continuous collision detection
* Collision shapes include: sphere, box, cylinder, cone, convex hull using GJK, non-convex and triangle mesh
* Soft body support: cloth, rope and deformable objects
* A rich set of rigid body and soft body constraints with constraint limits and motors
* Optional optimizations for PlayStation 3 Cell SPU, CUDA and OpenCL.

Apart from the features mentioned above, the physics community hosts an Active Physics Forum for general discussion around simulation for Games and Animation. These forums provided our team with valuable insights whenever we ran into issues with game development.

## 1.3 Sound Engine

Sound feedback is quintessential for any decent gaming experience, and for a typical C++ application there are many open source third party libraries that provide a shim layer to the audio driver. On Linux, the framework is the popular ALSA project (Advanced Linux Sound Architecture), and for Windows, the Core Audio APIs. Such low-level driver system calls implemented in the APIs are heavily hardware and platform dependent, and as such have no common abstraction to make the developer’s work easy. Fortunately, there exists an open source library called IrrKlang that does this very role of providing the abstract routines to invoke across all platforms, but has its own platform-specific implementation and under-the-hood library routine definitions.

The sound engine under the IrrKlang namespace has access to playing system-wide files of the standard .ogg, .wav, and .mp3 file formats. Since wave formats are standard, and do not require a plugin, they are the ideal sound sources to use in a cross-platform application. Moreover, IrrKlang provides methods to add custom sound effects, edit playback information, and handle events – capabilities aptly suited for game development. With the sound engine in place, event-based audio playback is active. An appropriate background music for a pacy game, audio effects for shots fired, engine revving are obtained, and set up wherever appropriate.

## 1.4 Thread-safe playback

For quick bursty sound effects such as gunfire and explosion, the playback is a simple API call to the play music interface of IrrKlang. Introducing a new thread to the already multithreaded bullet library, requires a basic mutual exclusion implementation. For a track like background music which loops infinitely, spawning a new thread is the right approach, while the thread literally executes in the background. Subsequently, other sound effects are invoked during runtime as and when events occur. For the added pause game functionality, an idle state of thread is communicated/polled across, and the game enters a pause state. Resetting the pause button wakes up the idle threads and resumes the game.

The threading mechanism is the POSIX thread library that provides multithreading during runtime with the pthread\_t type class for handling threads. Mutual exclusion is accomplished by the semaphore portion of the same pthreads library. Fortunately, this is available for both Windows and GNU-based systems, although the set up process of Windows is much more tedious than Linux since the only addition is a linker flag during compilation.

# 2. Game Details

## 2.1 Controls/Interface

Users will be shown all the keys allotted to control the movement of vehicle on the upper left corner of the screen by default. The user has the freedom to toggle the display using “F8”.

The basic car controls are:

* Up Arrow Key
  1. Allows the user to Increase the speed of the car
* Down Arrow Key
  1. Allows the user to decrease the speed of the car
* Left Arrow Key
  1. Allows the user to steer the car to the Left
* Right Arrow Key
  1. Allows the user to steer the car to the right
* Mouse Pointer
  1. Allows the user to aim at its target
* Right Click
  1. Allows the user to shoot bullets at the target being aimed at
* Space Bar
  1. Allows user to reset the car to original position
* P
  1. Allows the user to pause the game
* Q
  1. Allows the user to quit the game
* F8
  1. Toggles the display of user controls

These controls are handled in:

* DemoApplication::mouseFunc() [Shooting Bullets]
* DemoApplication::keyboardCallback() [Pause and Quit the Game]
* ForkLiftDemo::specialKeyboardUp() [Car movement controls ↑,←,→,↓,Space Bar]

## 2.2 Car

The basic car constitutes of car body (chassis), 2 front wheels and 2 back wheels and a canon fitted on top of chassis. The chassis is constructed with btRigidBody-box and canon is constructed with btRigidBody-Cylinder. The canon is attached to the Chassis with help of soft restraints so that it can give an effect of recoil on shooting a bullet. Both are integrated in an object ‘vehicle’ which is a helper class to set many car properties like engine capacity, mass, max speed etc. An object ‘wheel’ is also created and 4 instances of it are bound to ‘vehicle’ using the API ‘AddWheel’. Below is a list of modifications performed on the car, bullets and gameplay depending on the destruction points earned

### 2.2.1 Glide Mode

On earning 20 points, the car gets an upgrade with glider wings. The wheels disappear and due to this there is no friction with the ground. Friction is simulated artificially in this case. The result of this is higher speed in glide mode and smoother controls.

### 2.2.2 Monster Truck Mode

On earning 40 points, the car modifies into a monster truck. In this avatar of car, the chassis, wheel size and the engine capacity increases significantly. This gives the vehicle sturdy movements and increased speed.

### 2.2.3 Harpoon as bullets

On earning 60 points, the bullets change in shape and mass and resembles harpoons instead of sphere balls.

### 2.2.4 30 seconds increase in total time

On earning 80 points, the total game time increases by 30 seconds. So now the game finishes at 150 seconds instead of default time of 120 seconds.

## 2.4 Audio

Audio playback, as described in section 1, is achieved by the extensive IrrKlang library. In particular, the classes ISound, ISoundSource and ISoundEngine provide the interface to the platform-dependent driver modules for both Windows and Linux. ISound handles real-time execution of the current track under play, ISoundSource is a means to document the current track source, and to override the source of sound played. The main connection to the hardware is the ISoundEngine object which holds a pointer to the default configuration of the audio device currently playing; this is set up by an API call to createIrrKlangDevice() method that does a bookkeeping operation on the current output device (speaker/headphone) and introduces the playback to the main thread. It returns a reference to the current ISoundEngine\* sound playing which becomes the handle for audio operations.

Obtaining the actual sounds needed to be done under licensed sources, and so they were obtained from the public domain space. The tracks of .wav formats were downloaded and placed in the media directory for the application to lookup. At application startup, a thread is spawned to begin playback of the background music; an 8-bit track was chosen to give a 1990s Nintendo DS feel to the gameplay, to the general approval of beta testers. Event callbacks such as car throttle, reverse, and gunfire and handled via glutKeyboardCallback methods which are then forwarded to the appropriate sound handler which plays the required sound.

Since the background music thread is spawned and the main thread continues with running the glutMainLoop(), waiting for the BG thread to finish will never occur because it runs on an infinite loop. The threads never call pthread\_join() and the application never quits. To fix this, a simple mutual exclusion solution was employed. The background thread spins on a mutex/lock that is either PLAYING (0) or QUIT (1), and the main thread sets this value. By placing this shared variable in a critical section and adding a semaphore variable for mutual exclusion, the lock value is set to QUIT by the main thread and since it is placed in a critical section, this triggers a thread-safe quit operation by the background thread, and all threads eventually end by the pthread\_join() barrier.

The actual playing of sound is a straightforward call from the ISoundEngine interface to the play2D() method which takes either the ISoundSource or the full path (relative path in Linux) in Windows, to the sound source. Since wave formats function across both platforms, they were chosen for consistent playback. It is essential that the window to the audio devices must be manually closed by the application developer. This is done by a de-allocation of all ISound and ISoundEngine objects via the drop() method invocation. While reading from file for real-time applications is computationally expensive, this mechanism is handled by IrrKlang’s under-the-hood library while loads the sound file to memory and handles buffering on its own. Thus, portions of longer tracks are either pre-fetched, or smaller tracks are played on the fly.

While car sounds are simulated, the amount of realism is restricted since legitimate audio sources were not on ready demand. Real development in today's gaming community is done with live recording of the engine revving sounds from testbeds that run the cars. But, the current implementation has a speed-based sound for each RPM values, and it simulates a near ideal sound effect. Thus, audio is integrated fully into the application code.

## 2.5 Game Play

When the game is loaded, the screen is displayed which consists of the car and the buildings. It takes approximately 8-10 seconds for the game to load for the user to start. In the top left corner the current score, speed of the car and time limit are displayed. In the top right corner the player controls are displayed. A mouse pointer shows where the bullets fired by the canon, which is on the car, are directed. The movement is controlled by the arrow keys i.e. top, down, left and right.

The objective is to move around and continuously destroy building blocks. A boundary is defined in which the car can move around. If the car exceeds or goes beyond the boundary, it falls into an infinite depth and the game must be restarted in order to play again. There is music running in the background while the game is being played. Also there are sounds of bullets being fired and engine revving up. As the player fires bullets at the buildings his score increases with collision with the blocks.

The game lasts for 2 minutes and based on how the player performs he/she will get subsequent upgrades. On reaching a score of 20 points, the car gets gliders which increases its speed which helps the user in faster navigation around the world. Once a score of 40 points is reached, the car evolves into a truck with bigger wheels and stronger movements. After reaching a score of 60, the bullets get replaced by harpoons which helps in taking down more building blocks at a particular instant with a single collision. If the player is able to reach a score of 80 before the time elapses they get a bonus time of 30 seconds.

Once the time limit is reached, bullets cannot be fired and score becomes static.

# 6. Acknowledgement

We would like to thank Dr. Philip Elton Amburn for introducing us to the world of computer graphics and providing the opportunity to work on the game. His suggestions have been vital in enhancing many aspects of the game and also guided us to approach the problems in a more methodical way. We would also like to thank our fellow students Shubham, Tyson, Shivkanth, Aditya and Pradeep for testing the game and providing feedback. Their constructive feedback helped us in enhancing certain features of the game which had a positive impact on the gameplay experience.

# 7. References

<http://bulletphysics.org/wordpress/>

<http://www.bulletphysics.org/mediawiki-1.5.8/index.php?title=Collision_Callbacks_and_Triggers>

<http://bulletphysics.org/Bullet/BulletFull/>

<http://bulletphysics.org/mediawiki-1.5.8/index.php/Collision_Detection>

<http://bulletphysics.org/Bullet/phpBB3/index.php?sid=ae22f5978d9352415423bc7628dfdeee>

<https://code.google.com/p/bullet/downloads/list>

<http://www.ambiera.com/irrklang/>

<http://www.ambiera.com/irrklang/docu/>

<http://en.wikipedia.org/wiki/Irrlicht_Engine>

<https://www.youtube.com/watch?v=J9HaT23b-xc>