Logo, company name

Description automatically generated

**CSCI 5306 – Applied Program Comprehension**

Instructor – Dr. Tami Meredith

Assignment – 2

***Crew – 11***

Benny Tharigopala – B00899629

Hardee Garala – B00869195

Heli Patel – B00868906

Parthkumar Patel – B00899800

Vishal Jaiswal – B00867181

Vishvesh Naik – B00885077

## Tools with solid fill Building SuperTuxKart on Linux

The first requirement to build the SuperTuxKart is resources which are going to be used for building the application. To obtain these resources, we have performed two commands in Linux terminal to retrieve the files that are needed and cloned into local directories in system. Here, we are using Xfce variant of Ubuntu in order to perform building task for the application.

The first command we have performed is given below:

*git clone https://github.com/supertuxkart/stk-code stk-code*

Text

Description automatically generated

Figure 1. Cloning of git repository

Git is version controlling tool that can host multiple versions of same software where various developers can fork the individual branch and push the changes to merge them later on for feature update. By using command line utility of “git clone”, we can fetch entire repo or individual files from certain git repo owners. Here, we have cloned the supertuxkart repo under the “stk-code” directory.

Another command we have performed after cloning the git repo is given below:

*svn co https://svn.code.sf.net/p/supertuxkart/code/stk-assests stk-assets*

A computer screen capture

Description automatically generated with medium confidence

Figure 2. Checking out a working copy from repository

SVN or Subversion is another software versioning system which allow user to maintain different versions of files. In order to copy the existing directory to local system, “svn co” command is used. Here, we have fetched assets required to build application from svn into “stk-assets” directory.

To continue the process of building the application, we are required to install some dependency packages into the system in order to perform the compilation task. Without these packages, the build process is not going to take place because of dependency conflict. Here is the list of packages that are required:

OpenAL (recommended: openal-soft-devel)

Ogg (libogg-dev)

Vorbis (libvorbis-dev)

Freetype (libfreetype6-dev)

Harfbuzz (libharfbuzz-dev)

libcurl (libcurl-devel)

libbluetooth (bluez-devel)

openssl (openssl-dev)

libpng (libpng-devel)

zlib (zlib-devel)

jpeg (libjpeg-turbo-devel)

SDL2 (libsdl2-devel)

We have executed the following command inside the terminal to install these all dependencies:

*sudo apt-get install build-essential cmake libbluetooth-dev libsdl2-dev \*

*libcurl4-openssl-dev libenet-dev libfreetype6-dev libharfbuzz-dev \*

*libjpeg-dev libogg-dev libopenal-dev libpng-dev \*

*libssl-dev libvorbis-dev libmbedtls-dev pkg-config zlib1g-dev*

Text

Description automatically generated

Text

Description automatically generated

Figure 3. Installation of dependency packages

Apt-get is built-in package manager for Debian based Linux systems to install, manage and remove packages also known as utilities. By executing the command with sudo, we gain the superuser authorization to download and install packages into the system. We have executed the command with number of utilities to be installed that are going to be required later on in building process.

Now that we have all required resources to carry on the building process, we have created the directory “cmake\_build” inside the “stk-code” directory. By moving into the newly created directory with help of “cd” command which stands for change directory, we have executed the following command:

*cmake . .*

Text

Description automatically generated

Text

Description automatically generated

Figure 4. Generating makefiles with cmake utility

cmake is open-source utility for building automation and packaging of software by using the method which is not compiler dependent. It does not actually build any system but rather the build files for system which will be later used by other native building environments like make, ninja, android studio, xcode and visual studio.

After successful completion of building makefiles, we now can compile them to build the application. For that, we will be using the make utility with following command:

*make -j$(nproc)*

Text

Description automatically generated

Figure 5. Building config files with make utility

make is utility to build and maintain the groups of software by automatically determining the necessary pieces of program to re-compile them. It is not limited to just programs but can be useful where some files need to be updated automatically whenever changes occur. There must be makefiles in order to make works by updating the object files which are made by compiling source codes.

-j$(nproc) argument with make contains two parts of different purpose. Here, -j option is used to assign number of jobs specified in order to make the execution of building process much efficient. Whereas, $(nproc) will print into terminal whatever the (nproc) will return. Nproc is small bulit-in Linux utility that returns the number of available processing units to be utilized. Combining all these chunks together, the argument stands for available processing unit assigned to make utility or building application.

At the end, after successful build, we get the “supertuxkart” file under /bin directory.

Text, chat or text message

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Figure 6. Successful completion of building supertuxkart application

In order to execute the runnable file “supertuxkart”, we entered following command inside the “bin” directory where application file is:

*./supertuxkart*

Text

Description automatically generated

Figure 7. Executing command for supertuxkart runnable

After the loading of all resources for application, system prompted with GUI window having SuperTuxKart application running inside indicating proper working of built application inside the system.

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

Figure 8. SuperTuxKart application window running on Linux

In conclusion, building process for the SuperTuxKart was straightforward in Linux based operating system without any difficulties faced meanwhile. There are very few steps required to perform with none of the tweaking inside the host system. First step is to fetch resources with “git” and “svn” tools. Then makefiles are generated using those resources with the help of “cmake” utility. After that, “make” tool is used to finally build the application with the makefiles generated from last step. Finally, execute the runnable file from terminal prompt the window of SuperTuxKart application.

## Building SuperTuxKart on windows

Graphical user interface, website

Description automatically generated

Figure 9: Overview of the SuperTuxKart code build process in a Windows Operating System

### Visual Studio

In order to compile and build the code for SuperTuxKart we need to download Visual Studio or any Integrated Development Environment (IDE) to access the solution file for SuperTuxKart and generate relevant artifacts. Note that programmers can use the “gcc” command for the same purpose.

### Source Code and Assets

In order to build the artifacts we require the source code and and the v

## Building SuperTuxKart on ios