PROJECT 1

ROCKET LAUNCH SIMULATION GAME

Andrew Reid East

CIS-5 - 40718

February 2, 2015

Table of Contents

Introduction

The Rocket Launch Simulation Game is a simple physics-based simulation that challenges the player to launch a rocket into low Earth orbit, while working under the tight physical constraints similar to an actual liftoff from Earth. The game was inspired by a discussion I read online about the complexities of lifting a large payload to orbit, since fuel accounts for the majority of a rocket's weight. I wanted to explore this concept for myself, and I did a little research on the equations behind NASA's rocket launches. This led to the idea for a simulation that would be adaptable into a game.

Based on more extensive research centered on the Ideal Rocket Equation, I determined that I can simulate the velocity and position of a rocket using algebraic equations, varied over time, with the input data restricted to the efficiency of the engine, the mass of the rocket, the force of the thrust, and the burn time of the engine. My resultant equations do not account for air resistance or the change in acceleration due to gravity as altitude changes, but when I plugged in values for the Saturn V rocket's first stage, the results were within 80% of the expected values. This was a triumphant moment, since I was able to piece together these equations without the knowledge of calculus or rocket physics!

Game Play and Rules

In this game, the player is presented with the scenario of a rocket that must successfully reach the altitude of the International Space Station at 340 km in low Earth orbit. The rocket has pre-determined mass, payload, and engine efficiency. The player chooses how much thrust the engines will produce and how long to burn the engines. The game will calculate how much fuel the engine needs per second from this, and then the total mass of the fuel. It then runs the simulation, showing a simple text-based graphic of the rocket launching and approaching space, while displaying the altitude and speed at each time interval. If the user's parameters did not result in a successful launch, they can modify their previous values to try again. The input loops until the user is victorious.

There is an easy mode where thrust is also pre-determined, and the player can practice with different times to burn the engine, getting a feel for how the rocket responds. There is also a hard mode where the player must position the rocket at the ISS' altitude very precisely at the apex of its ballistic flight after the burn is done so it moving slowly enough to dock.

The game does not keep score or having anything to "beat". The challenge of this game is that there is a narrow range of values for the magnitude of the thrust and length of the burn that will propel the simulated rocket all the way into space while not loading the rocket with so much fuel that it cannot lift off at all. As it is, the extent of the game is to play with the physics and figure it out for yourself; more of a sandbox than a true game. Happy flying!

Development Summary

Lines of Code Metrics

Total Lines of Code: 412
Physical Executable Lines of Code: 287
Logical Executable Lines of Code: 178
Number of Methods: 9
Number of Variables: 42

Programming Methods

To develop this game, I used bottom-up approach to solve the problems inherent in the development. The hardest part for me to overcome was the basic physics calculations, so I began there. I had to learn the basics of powered rocket flight, and then find equations that both satisfactorily simulated the actual motion of rockets and were simple enough for me to understand. I found a good resource on NASA's Glen Research Center Exploration web site (NASA), but the most applicable source was from course documentation from an MIT course on Thermodynamics and Propulsion by Prof. Z. S. Spakovszky (Spakovszky). Through that resource, I got equations for altitude and velocity that depended on time, and the professor had reduced the terms that needed to be integrated into their lower order terms. I just had to use my brief understanding of the rocket equation to plug in values from an actual rocket—I chose the Saturn V, with vital statistics taken from Wikipedia (Saturn V).

With an equation and physical parameters to test with, I starting testing using C++. I converted the equations to code, used set values, and compared the results to what I calculated by hand. After correcting some bugs (such as converting meters to hectometers rather than kilometers, and wondering why I was off by a whole order of magnitude!), I got results both on paper and from my code that were similar to those presented on the Saturn V reference page. My results showed my simulated rocket going higher and faster than real life, which I attributed to neglecting to calculate air resistance slowing it down.

Height of Rocket during Burn

$$h_b = g\left[-t_b I_{sp}\frac{\ln\left(\frac{m_{v0}}{m_v}\right)}{\left(\frac{m_{v0}}{m_v}-1\right)} + t_b I_{sp} - \frac{1}{2}t_b^2\right]$$

Velocity of Rocket during Burn

$$v = g\left[I_{sp}\ln\left(\frac{m_{v0}}{m_v}\right) - t\right]$$

Next, I put my equations into a loop that ran time from t=0 the moment when all fuel has been consumed. All the physical parameters were changed to variables, with the consideration that they would eventually be attributes of a function. I set it up to output velocity, altitude, and fuel at each second, and ran the simulation. Once I was satisfied the rocket was flying smoothly, I created an ASCII graphics display that showed the rocket flying from the launch pad to its goal, adding a frame of animation for each second passed. Lastly, I added in the capability for the rocket to enter free flight after the fuel was burnt, using the simple ballistics without air resistance equations based on velocity at the end of burn and acceleration from gravity.

With the physics completed, I built a user interface and game structure around the simulation. I wrapped the loop that ran the simulation over time inside of a function, taking parameters for the physical properties of the rocket (predefined and variable), running and displaying the equation, and reporting back to the caller if the flight was a success or a crash. Then, I build another function to run the game, taking input from the console, and letting the user repeat the input until they are successful in launching the rocket. This function asks for one or two variables and has harder win conditions based on the difficulty passed to it. Finally, my main function ask the user to choose a difficulty or quit the game, and simply calls the game function based on that choice.

The only thing I had left to do was testing, searching out bugs in the input routines and making the game more usable. One of the biggest compromises I made was changing some of the physical parameters, deviating from the Saturn V rocket I was using as a model. I had to tweak the engine efficiency to widen the range of values acceptable for thrust and burn time so that the rocket would actually lift off the launch pad and fly more than a few hundred meters up. I also chose to speed up the simulation, first by a factor of four, and then deciding that each second passing in the simulation was 1/100 of real time. This made it so each time the player experiments with a new test run, it is not laborious to watch the rocket fly.

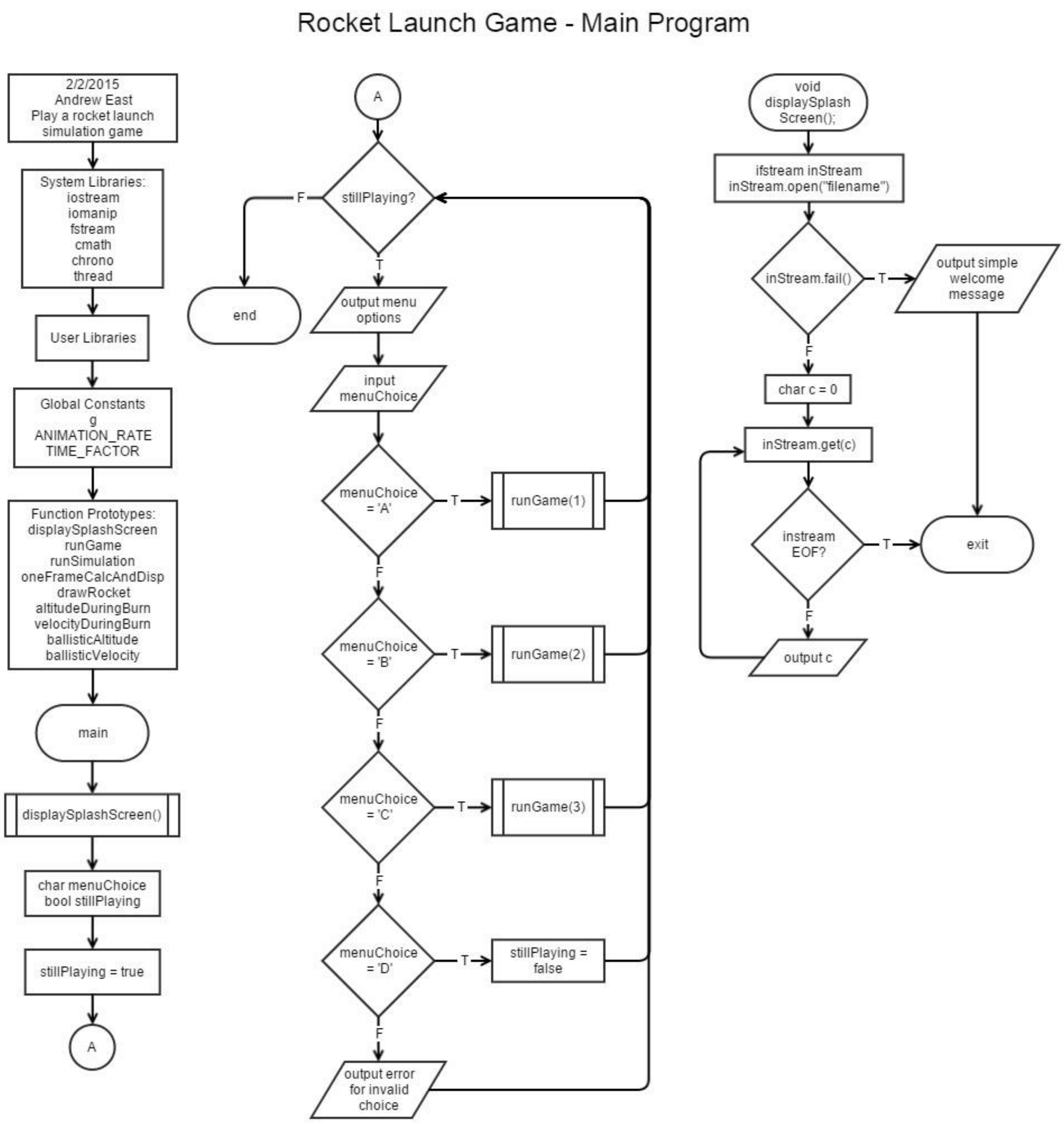
My final estimation for effort applied to this project is 16 hours coding and testing the program. I estimate I needed 12 hours of prep work, combining researching the physics of rocketry, finding the correct equations to use in a simplified physics model, doing the math to rearrange the equations to give me the desired output, and testing those equations that they give the correct results.

Specifications

Major Methods

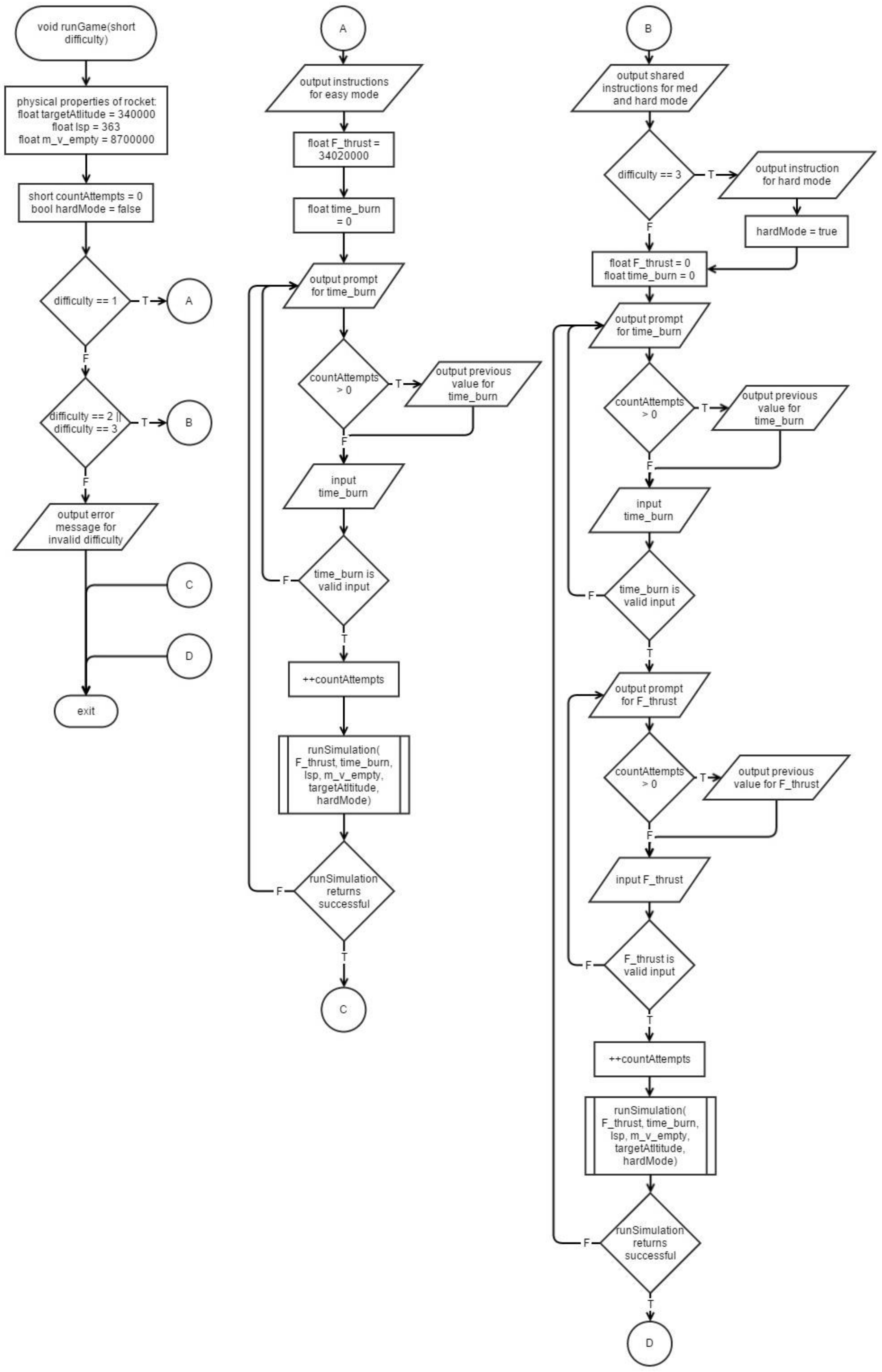
- `void displaySplashScreen()`
  - Reads an ASCII art from a text file, and outputs it to the console
- `void runGame(short difficulty)`
  - Lets a user input guesses for the rocket, and then runs the simulation for those guesses. Loops until the user is successful. First, sets up the physical parameters for the rocket.
- `bool runSimulation(float F_thrust, float time_burn, float Isp, float m_v_empty, float targetAltitude, bool hardMode, float targetVelocityThreshold = 100)`
  - Runs a complete simulation, from launch time to when the rocket either reaches its goal or crashes back to earth. Returns true if the launch was a success.
- `bool oneFrameCalculateAndDisplay(float& currentAltitude, float& currentVelocity, float t, float Isp, float time_burn, float m_v0, float mdot, float targetAltitude)`
  - Accepts the physical parameters of the rocket and the time desired, and calculates position and velocity at that time. Also, displays that data and the corresponding frame of animation. Returns true if the rocket is either past its goal or crashed.
- `void drawRocket(float y, float ceiling, unsigned short cols = 80)`
  - Displays a single frame of animation.

Flowcharts

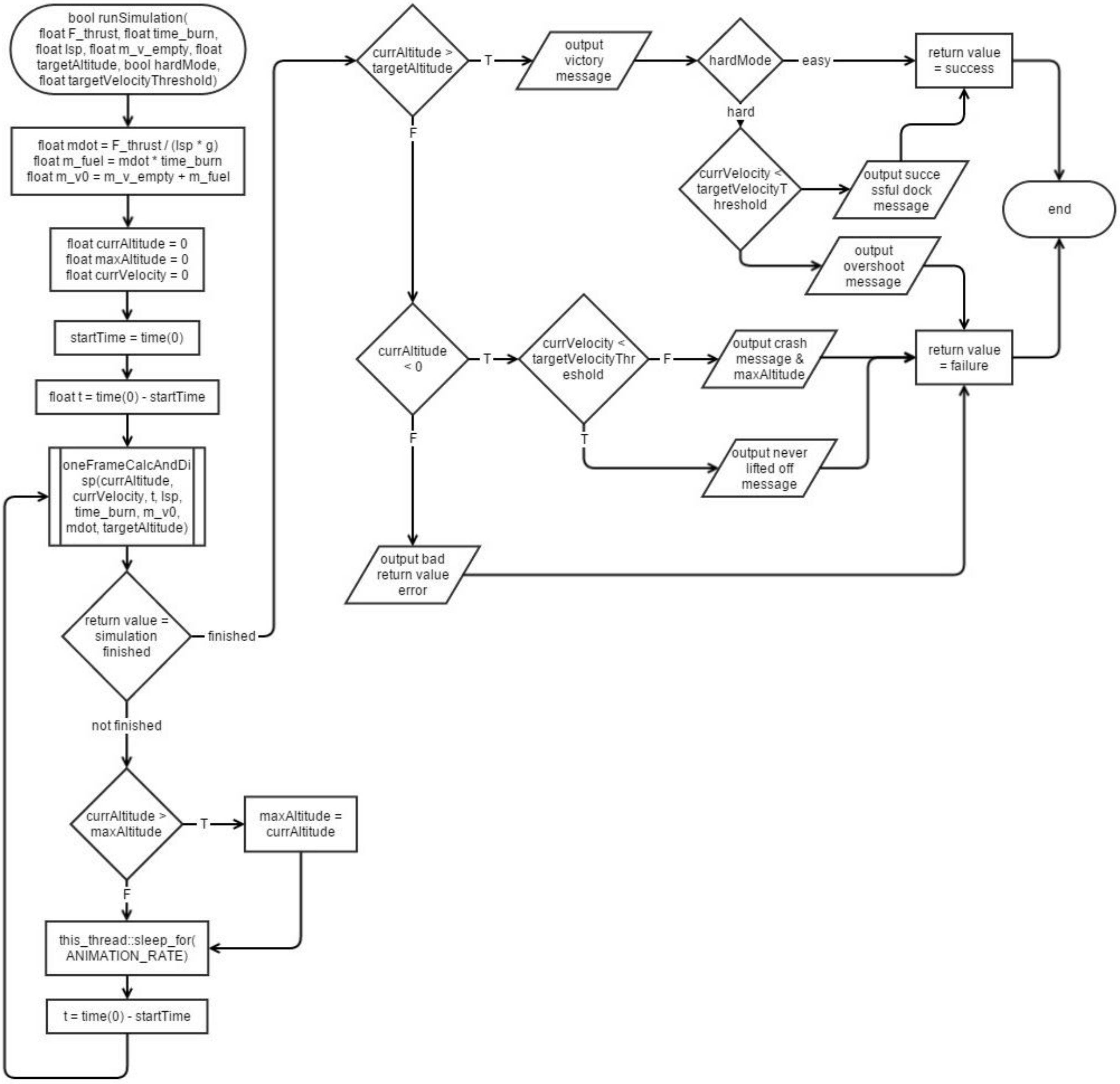


Rocket Launch Game - Main Program
2/2/2015
Andrew East
Play a rocket launch simulation game
System Libraries:
iostream
iomanip
fstream
cmath
chrono
thread
User Libraries
Global Constants
g
ANIMATION_RATE
TIME_FACTOR
Function Prototypes:
displaySplashScreen
runGame
runSimulation
oneFrameCalcAndDisp
drawRocket
altitudeDuringBurn
velocityDuringBurn
ballisticAltitude
ballisticVelocity
main
displaySplashScreen()
char menuChoice
bool stillPlaying
stillPlaying = true
A
A
stillPlaying?
F
end
T
output menu options
input menuChoice
menuChoice = 'A'
T
runGame(1)
F
menuChoice = 'B'
T
runGame(2)
F
menuChoice = 'C'
T
runGame(3)
F
menuChoice = 'D'
T
stillPlaying = false
F
output error for invalid choice
void displaySplash Screen();
ifstream inStream
inStream.open("filename")
inStream.fail()
T
output simple welcome message
F
char c = 0
inStream.get(c)
instream EOF?
T
exit
F
output c

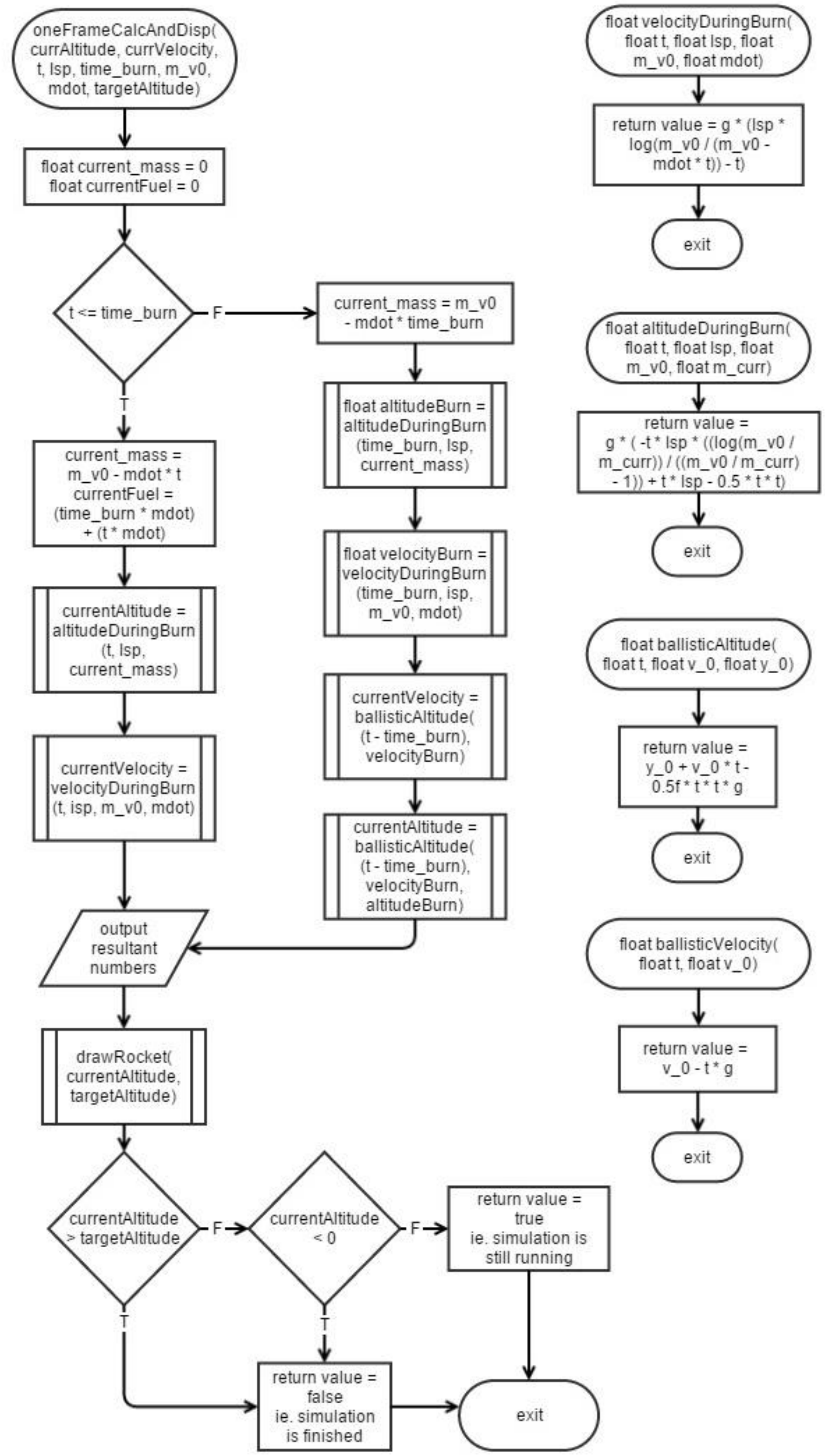
Rocket Launch Game - runGame Subprogram


void runGame(short difficulty)
physical properties of rocket: float targetAtlitude = 340000 float Isp = 363 float m_v_empty = 8700000
short countAttempts = 0 bool hardMode = false
difficulty == 1
T
A
F
difficulty == 2 || difficulty == 3
T
B
F
output error message for invalid difficulty
C
D
exit
A
output instructions for easy mode
float F_thrust = 34020000
float time_burn = 0
output prompt for time_burn
countAttempts > 0
T
output previous value for time_burn
F
input time_burn
time_burn is valid input
F
T
++countAttempts
runSimulation(F_thrust, time_burn, Isp, m_v_empty, targetAtltitude, hardMode)
runSimulation returns successful
F
T
C
B
output shared instructions for med and hard mode
difficulty == 3
T
output instruction for hard mode
hardMode = true
F
float F_thrust = 0 float time_burn = 0
output prompt for time_burn
countAttempts > 0
T
output previous value for time_burn
F
input time_burn
time_burn is valid input
F
T
output prompt for F_thrust
countAttempts > 0
T
output previous value for F_thrust
F
input F_thrust
F_thrust is valid input
F
T
++countAttempts
runSimulation(F_thrust, time_burn, Isp, m_v_empty, targetAtltitude, hardMode)
runSimulation returns successful
F
T
D

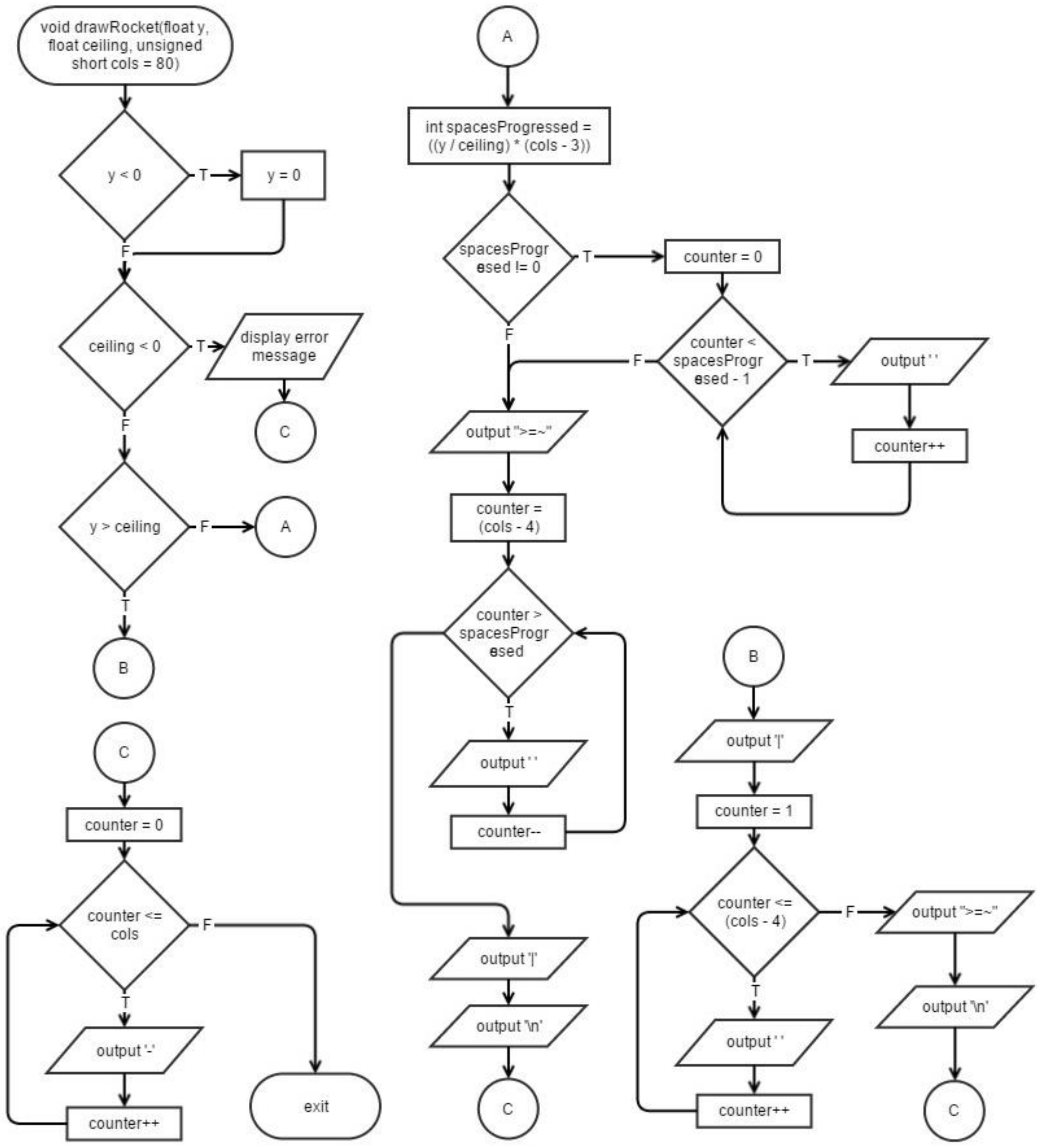
Rocket Launch - runSimulation Subprogram


bool runSimulation(float F_thrust, float time_burn, float Isp, float m_v_empty, float targetAltitude, bool hardMode, float targetVelocityThreshold)
float mdot = F_thrust / (Isp * g)
float m_fuel = mdot * time_burn
float m_v0 = m_v_empty + m_fuel
float currAltitude = 0
float maxAltitude = 0
float currVelocity = 0
startTime = time(0)
float t = time(0) - startTime
oneFrameCalcAndDisp(currAltitude, currVelocity, t, Isp, time_burn, m_v0, mdot, targetAltitude)
return value = simulation finished
finished
not finished
currAltitude > maxAltitude
T
F
maxAltitude = currAltitude
this_thread::sleep_for(ANIMATION_RATE)
t = time(0) - startTime
currAltitude > targetAltitude
T
F
output victory message
hardMode
easy
hard
return value = success
currVelocity < targetVelocityThreshold
output successful dock message
output overshoot message
end
currAltitude < 0
T
F
currVelocity < targetVelocityThreshold
F
T
output crash message & maxAltitude
output never lifted off message
return value = failure
output bad return value error

RocketLaunch - oneFrameCalcAndDisp Subprogram


oneFrameCalcAndDisp(currAltitude, currVelocity, t, Isp, time_burn, m_v0, mdot, targetAltitude)
float current_mass = 0
float currentFuel = 0
t <= time_burn
F
T
current_mass = m_v0 - mdot * time_burn
float altitudeBurn = altitudeDuringBurn (time_burn, Isp, current_mass)
float velocityBurn = velocityDuringBurn (time_burn, isp, m_v0, mdot)
currentVelocity = ballisticAltitude((t - time_burn), velocityBurn)
currentAltitude = ballisticAltitude((t - time_burn), velocityBurn, altitudeBurn)
current_mass = m_v0 - mdot * t
currentFuel = (time_burn * mdot) + (t * mdot)
currentAltitude = altitudeDuringBurn (t, Isp, current_mass)
currentVelocity = velocityDuringBurn (t, isp, m_v0, mdot)
output resultant numbers
drawRocket(currentAltitude, targetAltitude)
currentAltitude > targetAltitude
F
T
currentAltitude < 0
F
T
return value = true ie. simulation is still running
return value = false ie. simulation is finished
exit
float velocityDuringBurn(float t, float Isp, float m_v0, float mdot)
return value = g * (Isp * log(m_v0 / (m_v0 - mdot * t)) - t)
exit
float altitudeDuringBurn(float t, float Isp, float m_v0, float m_curr)
return value = g * (-t * Isp * ((log(m_v0 / m_curr)) / ((m_v0 / m_curr) - 1)) + t * Isp - 0.5 * t * t)
exit
float ballisticAltitude(float t, float v_0, float y_0)
return value = y_0 + v_0 * t - 0.5f * t * t * g
exit
float ballisticVelocity(float t, float v_0)
return value = v_0 - t * g
exit

Rocket Launch Game - drawRocket Subprogram



C++ Programming Concepts

Variables	`float F_thrust = 0.0f;`
Primitive Data Types	`char menuChoice = 0;`
	`const int TIME_FACTOR = 100`
	`unsigned short i;`
	`bool hardMode = true;`
	`float F_thrust = 3.402e7;`
	double not used because precision not needed
Constants	`const float g = 9.80665; // m/s^2`
Console Input and Output	`cout << "You are launching a large single-stage rocket to the International Space Station";`
	`cin >> F_thrust;`
if	`if (currAltitude > maxAltitude)`
	`    maxAltitude = currAltitude;`
if, else if, else	`if (currAltitude > targetAltitude)`
	`  if (hardMode && currVelocity <= targetVelocityThreshold)`
	`  else if (hardMode)`
	`  else //not hardMode`
	`else if (currAltitude < 0)`
	`  if (abs(currVelocity) <= targetVelocityThreshold)`
	`  else`
	`else`
switch	`switch (menuChoice)`
	`{`
	`  case 'A': case 'a':`
	`    runGame(1);`
	`    break;`
	`  case 'B': case 'b':`
	`    runGame(2);`
	`    break;`
	`  case 'C': case 'c':`
	`    runGame(3);`
	`    break;`
	`  case 'D': case 'd':`
	`    stillPlaying = false;`
	`    break;`
	`  default:`
	`    cout << "That was not a valid menu choice!" << endl;`
	`}`
while	`while (oneFrameCalculateAndDisplay(currAltitude, currVelocity, t, Isp, time_burn, m_v0, mdot, targetAltitude))`
do...while	`do`
	`{`
	`  cout << "Please input the burn time (in seconds): ";`
	`  cin >> time_burn;`
	`} while (time_burn < 0);`
for	`for (i = 1; i <= cols; ++i)`
	`  cout << '.';`
Boolean logic statements	`if (hardMode && currVelocity <= targetVelocityThreshold)`
Increment/decrement	`++countAttempts;`

Comments	`//run one stage of the animation, which returns false if the simulation has completed`
functions	`float velocityDuringBurn(float t, float Isp, float m_v0, float mdot);`
functions return a Boolean value	`bool runSimulation(float F_thrust, float time_burn, float Isp, float m_v_empty, float targetAltitude, bool hardMode, float targetVelocityThreshold = 100);`
void functions	`void drawRocket(float y, float ceiling, unsigned short cols = 80);`
call-by-reference parameters	`bool oneFrameCalculateAndDisplay(float& currentAltitude, float& currentVelocity, float t, float Isp, float time_burn, float m_v0, float mdot, float targetAltitude);`

File I/O	`ifstream fileStreamGraphic;`
	`fileStreamGraphic.open("splashScreen.graphic");`
	`if (fileStreamGraphic.fail())`
	`//display file char-by-char`
	`char c = 0;`
	`while (fileStreamGraphic.get(c))`
	`  cout << c;`

Sample Inputs and Output

```
**********************************************************
**                                                      **
**\   Welcome to the Rocket Launch Simulation Game!     **
***\                                                 *  **
****\                                         *         **
****)                                             *     **
****|)    ----    ---   --  -  -   >=-         *      *  **
****)                                            *      **
**********************************************************
What difficulty would you like to play at?
 A. Easy (one variable)
 B. Medium (two variables)
 C. Hard (two variables and must approach the station at low velocity)
 D. Exit program
A

You are launching a large single-stage rocket to the International Space Station
in low Earth orbit, 340.00 km. The scientists have already determined the
optimum thrust for the engine. You must figure out how long to burn the engine.
Longer burn will add weight to the rocket from fuel. Good luck!

Please input the burn time (in seconds): 165

   0.00 sec, Altitude:   0.00 km, Velocity:     0.00 m/s, Fuel Remaining: 100.00%
>=-                                                                            |
................................................................................

   6.22 sec, Altitude:   0.08 km, Velocity:    26.54 m/s, Fuel Remaining: 96.23%
```

```
>=-                                                                            |
................................................................................

Output Truncated
155.92 sec, Altitude: 101.19 km, Velocity:  1813.59 m/s, Fuel Remaining:  5.50%
|                      >=-                                                     |
................................................................................

 162.18 sec, Altitude: 113.06 km, Velocity:  1981.92 m/s, Fuel Remaining:  1.71%
|                         >=-                                                  |
................................................................................

 168.41 sec, Altitude: 125.74 km, Velocity:  2029.58 m/s, BURN COMPLETED:  0.00%
|                            >=-                                               |
................................................................................

Output Truncated

643.05 sec, Altitude: -15.59 km, Velocity: -2625.10 m/s, BURN COMPLETED:  0.00%
>=-                                                                            |
................................................................................

You have crashed! Phooey...
The highest altitude your rocket reached was 335.75 km.

Your last unsuccessful burn time was 165.00 s.
Please input the burn time (in seconds): 170

Output Truncated

362.11 sec, Altitude: 341.25 km, Velocity:   198.84 m/s, BURN COMPLETED:  0.00%
|                                                                           >=-
................................................................................

You have made it to the target altitude. Space, ahoy!
You were successful in 2 tries.


What difficulty would you like to play at?
 A. Easy (one variable)
 B. Medium (two variables)
 C. Hard (two variables and must approach the station at low velocity)
 D. Exit program
D

Thanks for playing!
```

References

NASA. "Idea Rocket Equation." n.d. *NASA's Glen Research Center.* Ed. Tom Benson. <http://exploration.grc.nasa.gov/education/rocket/rktpow.html>.

"Saturn V." n.d. *Wikipedia.* <http://en.wikipedia.org/wiki/Saturn_V>.

Sleep for milliseconds. n.d. <http://stackoverflow.com/questions/4184468/sleep-for-milliseconds>.

Spakovszky, Z. S. "The Rocket Equation." 2007. *Thermodynamics and Propulsion on MIT.edu.* <http://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node103.html>.

std::chrono::duration. n.d. <http://www.cplusplus.com/reference/chrono/duration/>.

Appendix: Code Listing

```
/*
    File:   main.cpp
    Author: Andrew Reid East
    Class: CSC-5 40718
    Created on January 13, 2015, 6:04 PM
    Purpose: Play a game where you try to launch a rocket to space by
guessing force exerted by the engine and length of time to burn the engine
 */

//System Libraries
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>

//<chrono> for simulation time keeping and <thread> for animation sleeping. I
referenced: http://stackoverflow.com/questions/4184468/sleep-for-milliseconds
//chrono needs C++11. To configure g++ in NetBeans, right click on Project.
Set Configuration -> Customize. Build -> C++ Compiler. C++ Standard == C++11.
#include <chrono>
#include <thread>
using namespace std;


//Global Constants
//Physics Constants
const float g = 9.80665; // m/s^2 - g_0 at sea level

//Game animation parameters: Tweak these to make the game run differently if
there is a lot of flicker on your machine
const int ANIMATION_RATE = 50; //in milliseconds
const int TIME_FACTOR = 100; //speed up simulation time by * 100 to make the
game feel less sluggish


//Function Prototypes

//displaySplashScreen: read an ASCII art from a file to display to <iostream>
void displaySplashScreen();
//preconditions:
//  "splashScreen.graphic" exists in the program directory

//runGame: Run an entire round of the game, taking inputs from <iostream>
until user is successful
void runGame(short difficulty);
//preconditions:
//  choice is a valid difficulty: 1, 2, or 3

bool runSimulation(float F_thrust, float time_burn, float Isp, float
m_v_empty, float targetAltitude, bool hardMode, float targetVelocityThreshold
= 100);
//preconditions:
```

```
// targetVelocityThreshold (100 m/s)

//oneFrameCalculateAndDisplay: make all physics calculations at time t and
display to <iostream>
bool oneFrameCalculateAndDisplay(float& currentAltitude, float&
currentVelocity, float t, float Isp, float time_burn, float m_v0, float mdot,
float targetAltitude);
//preconditions:
//  currentAltitude, currentAltitude - initialized variables to hold
calculated values
//  t (seconds) - time passed since launch. all calculations will be based
around this
//  Isp (seconds) - specific impulse in seconds
//  time_burn (seconds) - how long the rocket will run its engines for
//  m_v0 (kg) - mass of vehicle at time 0, still full of fuel
//  mdot (kg/s) - flow rate of fuel, how much fuel is burned per second
//  targetAltitude (m) - altitude where a rocket launch will "win"
//postconditions:
//  returns true if the rocket has either reached the targetAltitude or
crashed back to zero altitude
//  sets currentAltitude (m) to where the rocket is calculated at time t
//  sets currentVelocity (m/s) to velocity of rocket at time t

//drawRocket: draw the position of a rocket from 0 to ceiling, and fit within
cols characters wide
void drawRocket(float y, float ceiling, unsigned short cols = 80);
//preconditions:
//  y and ceiling are positive


//Physics Calculations:
//altitudeDuringBurn: get altitude at any time t during the rocket engine
float altitudeDuringBurn(float t, float Isp, float m_v0, float m_curr);
//velocityDuringBurn: get velocity at any time t during the burn
float velocityDuringBurn(float t, float Isp, float m_v0, float mdot);
//ballisticAltitude: get altitude of any object with no forces applied
besides gravity at time t
float ballisticAltitude(float t, float v_0, float y_0);
//ballisticVelocity: get velocity of any object with no forces applied
besides gravity at time t
float ballisticVelocity(float t, float v_0);


//Execution Begins Here
int main(int argc, char** argv)
{
  cout << fixed << setprecision(2);

  displaySplashScreen();

  char menuChoice = 0;
  bool stillPlaying = true;
  while (stillPlaying)
  {
    menuChoice = 0;
    cout << "What difficulty would you like to play at?" << endl
```

```
         << " A. Easy (one variable)" << endl
         << " B. Medium (two variables)" << endl
         << " C. Hard (two variables and must approach the station at low
velocity)" << endl
         << " D. Exit program" << endl;
    cin >> menuChoice;
    cout << endl;
    switch (menuChoice)
    {
      case 'A': case 'a':
        runGame(1);
        break;
      case 'B': case 'b':
        runGame(2);
        break;
      case 'C': case 'c':
        runGame(3);
        break;
      case 'D': case 'd':
        cout << "Thanks for playing!" << endl;
        stillPlaying = false;
        break;
      default:
        cout << "That was not a valid menu choice! Please enter A, B, C, or
D." << endl;
    }
  }


  return 0;
}


void displaySplashScreen()
{
  ifstream fileStreamGraphic;
  fileStreamGraphic.open("splashScreen.graphic");
  if (fileStreamGraphic.fail())
  {
    //display welcome message even if the file was not opened
    cout << endl << "Welcome to the Rocket Launch Simulation Game!" << endl
<< endl;
  }
  else
  {
    //display file char-by-char
    char c = 0;
    while (fileStreamGraphic.get(c))
      cout << c;
  }
}


void runGame(short difficulty)
{
  //340 km for the ISS:
  float targetAltitude = 340000;
  //rocket parameters (based on the Saturn V first stage):
  // float Isp = 263; // specific impulse of Saturn V first stage = 263
seconds. The Saturn V first stage cannot reach the ISS.
  float Isp = 363; // Modified efficiency of engine to be closer to the space
shuttle main engine (366 seconds)
  float m_v_empty = 870000; //total mass for rocket and payload minus stage 1
fuel

  short countAttempts = 0;
  bool hardMode = false;

  if (difficulty == 1)
  {
    cout << "You are launching a large single-stage rocket to the
International Space Station" << endl
         << "in low Earth orbit, " << (targetAltitude / 1000) << " km. The
scientists have already determined the" << endl
         << "optimum thrust for the engine. You must figure out how long to
burn the engine." << endl
         << "Longer burn will add weight to the rocket from fuel. Good luck!"
<< endl;

    //run simulation
    float F_thrust = 3.402e7; //actual thrust of the Saturn V first stage in
Newtons
    float time_burn = 0.0f;

    //get input and loop through simulation until user is successful
    do
    {
      do
      {
        cout << endl;
        if (countAttempts > 0)
          cout << "Your last unsuccessful burn time was " << time_burn << "
s." << endl;
        cout << "Please input the burn time (in seconds): ";
        cin >> time_burn;
      } while (time_burn < 0);
      ++countAttempts;
      cout << endl;
    } while (runSimulation(F_thrust, time_burn, Isp, m_v_empty,
targetAltitude, hardMode));
  }
  else if (difficulty == 2 || difficulty == 3)
  {

    cout << "You are launching a large single-stage rocket to the
International Space Station" << endl
         << "in low Earth orbit, " << (targetAltitude / 1000) << " km. You
must figure out the force from the engine and" << endl
         << "how long to burn the engine. Larger thrust and longer burn will
add weight to" << endl
         << "the rocket from fuel.";

    float targetVelocityThreshold = 100; //for hard mode
    if (difficulty == 3)
    {
      cout << " And, you must reach the space station traveling" << endl
           << "slower than " << (targetVelocityThreshold) << " m/s to dock, 
or you will shoot off into space. Tricky!" << endl;
      hardMode = true;
    }
    else
      cout << " Good luck!" << endl;

    //run simulation
    float F_thrust = 0.0f;
    float time_burn = 0.0f;

    //get input and loop through simulation until user is successful
    do
    {
      do
      {
        cout << endl;
        if (countAttempts > 0)
          cout << "Your last unsuccessful burn time was " << time_burn << " 
s." << endl;
        cout << "Please input the burn time (in seconds): ";
        cin >> time_burn;
      } while (time_burn < 0);
      do
      {
        cout << endl;
        if (countAttempts > 0)
          cout << "Your last unsuccessful engine thrust was " << (F_thrust) 
<< " kN." << endl;
        cout << "Please input the force of the engine thrust (in kN): ";
        cin >> F_thrust;
      } while (F_thrust < 0);
      ++countAttempts;
      cout << endl;
    } while (runSimulation(F_thrust * 1000, time_burn, Isp, m_v_empty, 
targetAltitude, hardMode));
  }
  else
  {
    cout << "Invalid difficulty for game: " << difficulty << endl;
    return;
  }

  cout << "You were successful in " << countAttempts << " tries." << endl << 
endl << endl;
}


bool runSimulation(float F_thrust, float time_burn, float Isp, float 
m_v_empty, float targetAltitude, bool hardMode, float 
targetVelocityThreshold)
{
  //derive properties from inputs:
```

```
  float mdot = F_thrust / (Isp * g); //substitution from: mdot = F_thrust /
v_e;
  float m_fuel = mdot * time_burn;
  float m_v0 = m_v_empty + m_fuel;

  float currAltitude = 0.0f;
  float maxAltitude = 0.0f;
  float currVelocity = 0.0f;

  //run simulation
  chrono::steady_clock::time_point startTime = chrono::steady_clock::now();
//C++11 time keeping. referenced:
http://www.cplusplus.com/reference/chrono/duration/
  float t = (chrono::duration<float>(chrono::steady_clock::now() -
startTime)).count() * TIME_FACTOR; //actual time adjusted by TIME_FACTOR to
speed up simulation

  //run one stage of the animation, which returns false if the simulation has
completed
  while (oneFrameCalculateAndDisplay(currAltitude, currVelocity, t, Isp,
time_burn, m_v0, mdot, targetAltitude))
  {
    if (currAltitude > maxAltitude)
      maxAltitude = currAltitude;
    //wait for ANIMATION_RATE to establish proper frame rate
    this_thread::sleep_for(chrono::milliseconds(ANIMATION_RATE));
//reference: http://stackoverflow.com/questions/4184468/sleep-for-
milliseconds

    //adjust t by actual time elapsed by computer clock, then multiply by
TIME_FACTOR
    t = (chrono::duration<float>(chrono::steady_clock::now() -
startTime)).count() * TIME_FACTOR;
  }
  if (currAltitude > maxAltitude)
    maxAltitude = currAltitude;

  //simulation is done and resultant parameters have been returned
  //check parameters vs. win conditions
  if (currAltitude > targetAltitude) //reached space
  {
    cout << "You have made it to the target altitude.";
    if (hardMode && currVelocity <= targetVelocityThreshold)
    {
      cout << endl << "And you're going slow enough to dock! Welcome to the
space station!" << endl;
      return false;
    }
    else if (hardMode)
    {
      cout << endl << "But you were going too fast and overshot your target!
Deep space is lonely..." << endl;
      return true;
    }
    else //not hardMode
    {
      cout << " Space, ahoy!" << endl;
      return false;
    }
  }
  else if (currAltitude < 0) //crashed back to the launch pad
  {
    if (abs(currVelocity) <= targetVelocityThreshold)
      cout << "Your rocket is not exerting enough force to overcome its mass
to lift off." << endl;
    else
      cout << "You have crashed! Phooey..." << endl;
    cout << "The highest altitude your rocket reached was " << (maxAltitude /
1000) << " km." << endl;
    return true;
  }
  else //(simulation calculators really shouldn't return this)
  {
    cout << "The rocket stopped somewhere hovering above the launch pad."
         << "This is a bug in either the game or physics itself."
         << "Please report this problem to the programmer or NASA,
appropriately." << endl;
    return true;
  }
}



bool oneFrameCalculateAndDisplay(float& currentAltitude, float&
currentVelocity, float t, float Isp, float time_burn, float m_v0, float mdot,
float targetAltitude)
{
  float current_mass_vehicle = 0.0f;
  float currentFuel = 0.0f;
  // changed to by-reference variable: float currentVelocity = 0.0f;
  // changed to by-reference variable: float currentAltitude = 0.0f;

  if (t <= time_burn) //during burn
  {
    current_mass_vehicle = m_v0 - mdot * t;
    currentFuel = (time_burn * mdot) - (t * mdot); //max - current
    currentAltitude = altitudeDuringBurn(t, Isp, m_v0, current_mass_vehicle);
    currentVelocity = velocityDuringBurn(t, Isp, m_v0, mdot);
  }
  else //after burn, add in ballistic flight
  {
    current_mass_vehicle = m_v0 - mdot * time_burn;
    float altAtEndOfBurn = altitudeDuringBurn(time_burn, Isp, m_v0,
current_mass_vehicle);
    float velocityAtEndOfBurn = velocityDuringBurn(time_burn, Isp, m_v0,
mdot);
    currentVelocity = ballisticVelocity((t - time_burn),
velocityAtEndOfBurn);
    currentAltitude = ballisticAltitude((t - time_burn), velocityAtEndOfBurn,
altAtEndOfBurn);
  }

  cout << setw(7) << t << " sec" << ", ";
  cout << "Altitude: " << setw(6) << (currentAltitude / 1000) << " km, ";
  cout << "Velocity: " << setw(8) << currentVelocity << " m/s" << ", ";
  cout << ((currentFuel > 0) ? "Fuel Remaining: " : "BURN COMPLETED: ") <<
setw(5) << (currentFuel / (mdot * time_burn) * 100) << "%" << endl;

  drawRocket(currentAltitude, targetAltitude);

  cout << endl;

  if (currentAltitude > targetAltitude || currentAltitude < 0)
    return false;
  else
    return true;
}

float velocityDuringBurn(float t, float Isp, float m_v0, float mdot)
{
  return g * (Isp * log(m_v0 / (m_v0 - mdot * t)) - t);
}
float altitudeDuringBurn(float t, float Isp, float m_v0, float m_curr)
{
  return g * ( -t * Isp * ((log(m_v0 / m_curr)) / ((m_v0 / m_curr) - 1)) + t
* Isp - 0.5 * t * t);
}
float ballisticAltitude(float t, float v_0, float y_0)
{
  return y_0 + v_0 * t - 0.5f * t * t * g; //unit check: m + m/s * s +
s*s*m/s^2
}
float ballisticVelocity(float t, float v_0)
{
  return v_0 - t * g; //unit check: m/s - s * m/s^2
}


void drawRocket(float y, float ceiling, unsigned short cols)
{
  unsigned short i;

  //a "crashed" rocket will be drawn on the launchpad
  if (y < 0)
  {
    y = 0;
  }

  if (ceiling < 0)
  {
    cout << "Cannot draw rocket. Target altitude cannot be negative." <<
endl;
  }
  else if (y > ceiling) //if progressed to the goal, just draw the ">" at the
ending goal mark
  {
    cout << '|';
    for (i = 1; i <= (cols - 4); ++i)
      cout << ' ';
    cout << ">=-";
    cout << endl;
  }
  else
  {
    unsigned short spacesProgressed = ((y / ceiling) * (cols - 3)); // y
percent of ceiling * number of characters total (minus 1 for final char)

    if (spacesProgressed != 0)
    {
      cout << '|';
      for (i = 0; i < (spacesProgressed - 1); ++i)
        cout << ' ';
      // cout << "i="<< i;
    }

    cout << ">=-";

    for (i = (cols - 4); i > spacesProgressed; --i)
    {
      cout << ' ';
    }
    cout << '|';// << endl;
    cout << endl;
  }

  //draw a line after
  for (i = 1; i <= cols; ++i)
    cout << '.';
  cout << endl;
}
```