ARES – Software Project – Game Demo

Nícolas Mocaiber

1.0

Mon Feb 10 2020

AresUnityDemo

Initializing

Assign target and bullet prefab when load scene for the first time.
Target1, target2 and target3 prefabs should be assigned to GameInspector's Game Controller script.
Bullet prefab should be assigned on Shot Control script on Player/Weapon_Base/CannonPivot/Cannon/FirePosition.

Command List

Vehicle Movement
'w' – Move Vehicle Forward
's' – Move Vehicle Backward
'a' – Turn vehicle to the left
'd' – Turn vehicle to the right

Weapon Movement
'i' – Cannon elevation upward
'k' – Cannon elevation downward
'j' – Cannon base rotation to the left
'l' – Cannon base rotation to the right

Other Inputs
'b' – Begin Game after server/client connection
"space" – Shoot bullet
'c' – Change Camera
'm' – Toggle mini map on/off

Class Documentation

BaseMovement

Weapon base rotation movement independent from the vehicle

Public Attributes

0) float **m_TurnSpeed** = 100f
   Base rotation speed
1) float **moveDeltaTime** = 0.05f
   Time which the object will stay on command from server

Private Member Functions

0) void **Awake** ()
   Get component from server TCPServer class
1) void **FixedUpdate** ()
   Do Turn()
2) void **Turn** ()
   Read input and do base rotation movement

Private Attributes

Values to simulate unity function input.getaxis using input from server

0) **TCPServer client**
1) float **currentTime** = 0
2) float **m_TurnInputValue**
3) float **previousVerticalTime** = 0

BulletBehaviour

Weapon base rotation movement independent from the vehicle

Public Attributes

0) bool **destroyTargetIfRollingOnFloor** = false
   Prevents target from being destroyed when it hits bullet rolling on floor
   If false, makes bullet destroy target only when hit on air.
   If true, target destruction will be possible even if bullet is rolling
   This was implemented instead of doing the bullet being destroyed when touch the ground
1) float **selfDestroyTime** = 10f
   Destroy itself after time set
2) float **shootPower** = 1300
   Force value applied on shot

Private Member Functions

0) void **Awake** ()
   Since object attached to script is a prefab, search for classes component upon on instantiation.
   Get rigidbody component and assign Basepoint and Firepoint objects for vector direction calculation

1) void **OnTriggerEnter** (Collider other)
   Detect target collision based on **destroyTargetIfRollingOnFloor** value
2) void **Start** ()
   Vector direction calculation: normalized (Firepoint position - Basepoint position) vector
   Add force on calculated direction
   Set self destruction time

Private Attributes

0) GameObject **BasePoint**
   Position used on vector direction calculation
1) GameObject **FirePoint**
   Position used on vector direction calculation
2) Rigidbody **rb**

CannonElevation

Cannon elevation movement script based on input from server

Public Member Functions

0) float **AngleValue** ()
   Convert unity angle value to values between -180 and 180 for the weapon elevation angle.

Public Attributes

0) float **m_ElevateSpeed** = 40f
   Weapon elevation speed
1) float **maxAngle** = 60
   Weapon maximum elevation angle. Can me adjusted from the editor.
2) float **minAngle** = -10
   Weapon minimum elevation angle. Can me adjusted from the editor.
3) float **moveDeltaTime** = 0.05f
   Time which the object will stay on command from server

Private Member Functions

0) void **Awake** ()
   Get component from server TCPServer class
1) void **Elevate** ()
   Read input and rotate object around Z axis.
2) void **FixedUpdate** ()
   Do Elevate()
3) void **OnGUI** ()
   Display elevate values on screen

Private Attributes

Values to simulate unity function input.getaxis using input from server.

0) **TCPServer client**
1) float **currentTime** = 0
2) float **m_ElevateInputValue**
3) float **previousVerticalTime** = 0

Static Private Attributes

0) const int **THRESHOLD** = 20
   Threshold to limit elevation angle

GameController

Set game dynamic based on server input

Public Attributes

0) GameObject **camera1**
   Attach and detach parent to player uppon server game initialization
1) bool **gameIsRunning** = false
   Game is running status
2) bool **hasGameBeenStarted** = false
   Tells if the game has been started at least once
3) GameObject **miniCamera**
   Minimap camera
4) int **numberOfTargets**
   Counts number of targets on scene
5) GameObject **player**
   Activate player uppon server game initialization
6) Text **startText**
   Time left text
7) GameObject **target1, target2, target3**
   Targets prefab
8) int **Targets**
   Number of targets that will be instantiated when game starts. Can be set from the editor
9) int **targetsHit**
   Targets hit count
10) float **time** = 120f
    Game countdown time
11) float **winnerTime**
    Records winner time when game ends

Private Member Functions

0) void **Awake** ()
   Get component from server TCPServer class
   Get component from shotControl class
1) void **CheckCameraChange** ()
   Check server input for camera change
2) void **CheckGameStart** ()
   Check server input and call StartGame()
3) int **CountTargets** ()
   Counts targets on scene
4) void **EndGame** ()
   Detach main camera from player, deactivate player, destroy all targets
   Reset shots fired getTimeOnce and gameIsRunning variables
   Centralize and change text
5) void **InstantiateTargets** ()
   Instantiate randonly one of the three target prefab available based on Targets variable set on editor

6) void **OnGUI** ()
    Display Shots fired, targets left data
    Start game, change camera, toggle minimap buttons
7) void **StartGame** ()
    Call InstantiateTargets (), set player active, set main camera parent, reset text anchor to screen corner, update target count and time left variable.
8) void **ToggleMiniMap** ()
    Activate and deactivate minimap
9) void **Update** ()
    Call CheckCameraChange(), CheckGameStart(), ToggleMiniMap()
    Update number of targets left and target hit data
    Update countdown and check for game end

Private Attributes

0) GameObject **camera2**
    Alternative camera available
1) **TCPServer client**
    TCPclient class
2) GameObject **clientObject**
    TCPclient object
3) bool **getTimeOnce** = true
    Get time only once when game ends
4) **ShotControl shotControlInspec**
    Get data from ShotControl class
5) float **timeLeft**
    Time left to game ending

PlayerMovement

Vehicle movement script

Public Attributes

0) float **m_Speed** = 12f
    Vehicle movement speed
1) float **m_TurnSpeed** = 180f
    Vehicle turn speed
2) float **moveDeltaTime** = 0.05f
    Time which the object will stay on command from server

Private Member Functions

0) void **Awake** ()
    Get component from server TCPServer class
1) void **FixedUpdate** ()
    Call Move() and Turn()
2) void **Move** ()
    Read input and do vehicle movement
3) void **Start** ()
    Ger rigidbody component
4) void **Turn** ()
    Read input and do vehicle turn movement

Private Attributes

Values to simulate unity function input.getaxis using input from server

0) **TCPServer client**
1) float **currentTime** = 0
2) float **m_MovementInputValue**
3) Rigidbody **m_Rigidbody**
4) float **m_TurnInputValue**
5) float **previousVerticalTime** = 0

ShotControl

Shoot command and control script
Attached to FirePosition object

Public Attributes

0) float **fireDelta** = 0.5F
   Shot fire rate
1) GameObject **projectile**
   Shot prefab
2) int **shotsFired** = 0
   Shots fired count

Private Member Functions

0) void **Awake** ()
   Get component from server TCPServer class
1) void **Update** ()
   Check server input and shot if within fire rate.

Private Attributes

0) **TCPServer client**
1) float **myTime** = 0.0F
2) GameObject **newProjectile**
3) float **nextFire** = 0.5F
4)

TargetMovement

Senoidal, horizontal and circular movement configuration. Set one of them randonly on start.

Public Attributes

0) float **maxAngularSpeed** = 4f
   Max angular speed for circular movement
   Changing this value also changes movement radius
1) float **maxFrequency** = 10f
   Max frequency for senoidal movement

2) float **maxHorizontalMoveSpeed** = 20f
   Max speed for horizontal movement
3) float **maxMagnitude** = 2f
   Max amplitude for senoidal movement
4) float **maxMoveSpeed** = 5f
   Max speed for senoidal movement
5) float **maxRotationRadius** = 5f
   Max radius for circular movement
   Changing this value also changes movement speed
6) float **minAngularSpeed** = 2f
   Min angular speed for circular movement
   Changing this value also changes movement radius
7) float **minFrequency** = 2.5f
   Min frequency for senoidal movement
   Changing this value also changes movement radius
8) float **minHorizontalMoveSpeed** = 10f
   Min speed for horizontal movement
9) float **minMagnitude** = 0.5f
   Min amplitude for senoidal movement
10) float **minMoveSpeed** = 3f
   Min speed for senoidal movement
11) float **minRotationRadius** = 2f
   Min radius for circular movement
   Changing this value also changes movement speed

Private Member Functions

0) void **Awake** ()
   Assign plane object
1) void **CircularMove** ()
   Set circular movement and change direction when hit walls
2) void **HorizontalHorizontalMove** ()
   Set horizontal movement angle directions and change it when hit walls
3) void **HorizontalVerticalMove** ()
   Set vertical movement angle directions and change it when hit walls
4) void **OnTriggerEnter** (Collider other)
   Check target colision with each specific wall
5) void **SenoidalHorizontalMove** ()
   Check senoidal horizontal direction and change it when hit wall
6) void **SenoidalMoveDown** ()
   Set senoidal movement downwards
7) void **SenoidalMoveLeft** ()
   Set senoidal movement to the left
8) void **SenoidalMoveRight** ()
   Set senoidal movement to the right
9) void **SenoidalMoveUp** ()
   Set senoidal movement upward
10) void **SenoidalVerticalMove** ()
   Check senoidal vertical direction and change it when hit wall
11) void **Start** ()
   Define which movement behaviour target will have
   Define a random instantiation position, if circular, reduces the available area to 85% of ground's size
   Set random direction values for all possible movements
   Set random attributes to all possible movement
   Chosen movement pattern will have random spawn position, random start direction and Set random values (speed, radius, frequency, magnitude, etc)

12) void **Update** ()
    Apply definied movement pattern

Private Attributes

0) float **angle** = 0f
    Angle of circular movement
1) bool **clockWiseCircleMovement**
    Circular movement direction
2) bool **horizontalPositiveMovement**
    Horizontal movement wall check
3) int **movementBehaviour**
    Set which movement pattern will be chosen
4) GameObject **plane**
    Get ground
5) Vector3 **planeSize**
    Get ground size
6) Vector3 **pos**
    Target Position
7) float **posX**
    X position used on circular movement calculation
8) float **posZ**
    Z position used on circular movement calculation
9) float **randomHorizontalIntensity**
    Set random horizontal direction
10) float **randomVerticalIntensity**
    Set random vertical direction
11) bool **verticalPositiveMovement**
    Vertical movement wall check

Static Private Attributes

0) const float **planeScaleConstant** = 5
    Ground scale is 5 times greater than usual scale

TCPServer

Creates TCP server and handle communication with c++ client application

Public Member Functions

0) string **getClientMessage** ()
    Message received from client
1) void **ReadInputOnlyOnce** (char input_a, char input_b, float currentTime, float previousTime, float deltaTime)
    Read server input only once if input equals to input_a or input_b. To read it only once, changes message received from client after deltaTime.
    currentTime and previousTime are used locally to store time values and compare to deltaTime
2) void **setClientMessage** (string message)
    If client is connected (isn't null), set message value

Public Attributes

0) float **base_input** = 0f
   Base value used to simulate unity function input.getaxis using input from server
1) float **elevate_input** = 0f
   Base value used to simulate unity function input.getaxis using input from server
2) float **h_input** = 0f
   Base value used to simulate unity function input.getaxis using input from server
3) float **v_input** = 0f
   Base value used to simulate unity function input.getaxis using input from server
4) GameObject **shotControlObject**
   Shotcontrol class object

Private Member Functions

0) void **Awake** ()
   Get component from shotControl class
   Get component from gameController class
1) void **FixedUpdate** ()
   Values to simulate unity function input.getaxis using input from server
2) String **GameStatus** ()
   Return a string with current game status when request is received
   Status = 's' means client is connected with server - this only will be displayed if game has never been started
   status = 'r' means the game has been started and is running
   status = 'e' means the game has ended after at least one start
   Gamestatus also return shots fired after status flag
   Gamestatus also return targets hit after flag 't'
   Example: r21t3 means that the game is running, 21 shots were fired and 3 targets have been hit
3) void **ListenForIncommingRequests** ()
   Runs in background TcpServerThread; Handles incomming TcpClient requests
4) void **SendMessage** (string messageSent)
   If client is connected (isn't null), send message to client
5) void **Start** ()
   Start TCP background thread

Private Attributes

0) string **clientMessage** = "0"
   Message to be sent to client
1) TcpClient **connectedTcpClient**
2) **GameController gameController**
3) string **gameStatus**
4) bool **getTimeOnce** = true
   Boolean to read time only once on ReadInputOnlyOnce()
5) int **hits**
   Target hit count
6) **ShotControl shotControlInspec**
7) int **shots**
   Shots count
8) TcpListener **tcpListener**
9) Thread **tcpListenerThread**

AresGameInput

Initializing

Run AresGameInput.cpp after unity's AresUnityDemo application is running, so that the TCP server will be already running before running AresGameInput TCP client.

The log files will be created based on the timestamp when the app is launched, in a way that there will never have a file with the same name or an already existing file will be overwritten.

Only valid input information will be stored, the rest will be ignored.

Class Documentation

LogMessage

```
#include "LogMessage.h"
```

Public Member Functions

- **LogMessage** ()
- **LogMessage** (char **input**)
- **~LogMessage** ()
- char **GetInput** ()
- string **GetTimeStamp** ()
- void **SetInput** (char s)

Private Attributes

- char **input**
- string **timestamp**

Friend Functions for operator overload

- std::ofstream & **operator**<< (std::ofstream &ofs, **LogMessage** &a)
  Operator overload to write object directly on file
- std::ostream & **operator**<< (std::ostream &os, **LogMessage** &a)
  Operator overload to use cout directly with object from class
- std::ifstream & **operator**>> (std::ifstream &ifs, **LogMessage** &a)
  Operator overload to read object directly on file

Constructor Documentation

LogMessage ()

Instantiate object with input set to '0' and calls GetActualTimeStamp() to set timestamp

LogMessage (char *input*)

Instantiate object with desired input and calls GetActualTimeStamp() to set timestamp

Functions

- bool **CheckValidInput** (char a)
  Return true if input is valid. New inputs need to be added in Vector validInput
- string **GetActualTimeStamp** ()
  Return current timestamp in the format "mm/dd/yyyy hh/mm/ss"
- string **GetInitialTimeStamp** ()
  Return timestamp in the format "mm_dd_yyyy_hh_mm_ss". Used on application setup to define log file's name
- void **GetTime** (int *hour, int *min, int *sec)
  Set pointer value for hour, min and sec based on current time
- int **HowManyShots** (string message)
  Return number of bullets shot based on server status message
  Number is taken from data between initial status char and 't' target flag on string
- int **HowManyTargetsHit** (string message)
  Return number of targets hit
  Number is taken from data between 't' target flag and end of string
- string **IntToString** (int a)
  Converts int to string. Used to retrieve timestamp data on desired format
- void **WriteLog** (string fileName, **LogMessage** log)
  Uses friend operator ovearload function to write object directly on log file

File Reference

AresGameInput.cpp

Divided in blocks, composed of the following:

- **Start Parameters**
- **Initialize WinSock**
- **Create socket**
- **Fill in a hint structure**
- **Connect to server**

- **While loop to send and receive data which consists of**:
  - Get current time and check if connection succeeded
  - Check if any key was pressed
  - Make sure the user has typed in something, print object and write it on file
  - Wait for response
  - Send status request to server
  - Read status message from server and identify game status,

If game has ended, identify shots fired and targets hit parameters, print and write on file precision, initial time, end time and duration time