CyberCIEGE UI Port to Unity: Work Plan

January 20, 2021

1 Overview

This document describes work necessary to create Unity UI elements for a revised CyberCIEGE game. Additional background and overall strategy can be found in CyberCIEGEPortingReport.pdf

CyberCIEGE has been split into two components: a simulation engine and a Unity based UI. The former manages all game play, network simulation, economy and game flow. The UI elements will mostly present the 3D environment and allow players to configure game objects via forms, menus, etc.

Existing 3D models and animations will be reused. These have been defined in Maya 8.

CyberCIEGE is a collection of independent scenarios, each of which is fully expressed by a Scenario Definition File (SDF). The intention is to resuse existing SDFs without modification. This goal affects the UI implementation of game geometry.

The initial goal is to create a functionally equivalent version of CyberCIEGE having a Unity UI in place of the DirectX based UI. The existing game can be obtained here:
`https://my.nps.edu/documents/107523844/108048740/setup-demo.exe`

2 General

The CyberCIEGE Unity UI classes interact with the simulation engine via the IPCManagerScript `SendRequest` method and per-UI methods called by the IPCManager. Most of the UI classes will present forms or other input schemes to allow users to change game state, e.g., add authorized users to a computer. In these simple cases, the UI class will package its revised state as XML and send that to the simulation engine via the IPCManager. Many of the Unity UI classes already include the desired data structures and methods for generating and sending XML. They lack useful UI elements for interacting with the user, having only crude menu stubs. Existing Unity stubs and management code is at
`https://github.com/mfthomps/CyberCIEGE-port-to-Unity3d`

With the exception of characters [1], the UI elements do nothing about their state other than reflect it in the forms/dialogs. Characters, switch animation loops and walk paths as described below under "Users".

A number of UI classes do not yet exist as Unity stubs, however they do exist as Java classes that provide forms and package XML suitable for sending via the IPCManager. These Java classes will need to be converted to functionally equivalent Unity classes. The specific Java classes that must be converted are identified in the UI Elements sections below. These Java UIs were created using NetBeans and are at:
`https://gitlab.nps.edu/mfthomps/cyberciege/tree/master/Ccse/CCdialogs`

A few of the Java UI elements are defined at:
`https://gitlab.nps.edu/mfthomps/cyberciege/tree/master/Ccse/CCSDT.`

NOTE however that most of the Java in that directory is only used for the Scenario Development Kit, which is not being ported as part of this effort.

The UI classes get their initial state from XML files generated by the simulation engine when a scenaro begins. Most of the UI classes already include this initialization.

While most UI class interaction with the simulation engine by sending state change requests via the SendRequest method, a handful also receive direction from the simulation engine via the IPCManager. These are "pushed" to the UI classes as seen in the Update method of the IPCManager.

[1] ...and tickers(scrollers), bubble speaks, fire and smoke...

The Unity UI has two scenes: a scenario selector described below under 4.9, and the scenario. A Unity stub that manages loading of a selected scenario is in GameLoadBehavior.cs

As previously noted, much of the XML generated by the UI elements and sent to the IPCManager is already defined either by Unity stubs or existing Java classes. Other gaps can usually be filled by referencing
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/ReplayLog.cpp`
which will consume all the XML for the simulation engine.

3 Environments and game space

3.1 Artwork

The floor is defined as an x/y grid. 3D office environments are placed on that grid. There are six office environments. An SDF will name the office envirnonments used in the scenario.

1. new big office (office5 / MS3OFFICE
2. new big military (office6 / MS3BASE)
3. old big office (mainoffice1 / MAINOFFICE1)
4. old big military (smalloffice / MAINOFFICE2)
5. home office
6. work office

Each scenario includes a main office and a small number ofoffsite offices (which may include the old big military).

Object naming is defined in
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/shape.cpp`
If it is not defined in shape.cpp, you can probably eignore it.

Maya art is in https://gitlab.nps.edu/mfthomps/cyberciege/tree/master/art/MayaSourceArt

Office selection/naming in
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/CreateOffice.cpp`
Msc objects, e.g., sofas, are placed on that grid depending on the main office selection, as defined in
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/InitWorld.cpp`
Mapping of game names for computers/devices to Maya files can be derived from
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/game/hardware.cpp`

3.2 Geometry

The new Unity geometry should be constructed to mimic the old CyberCIEGE geometry. The goal is to run existing scenarios without modification. Scenarios include predefined camera positions.

3.3 Workspace

The SDF includes a list of (index, x, y, N/S/E/W, furniture) tuples that define workspaces. The `furniture` values is desk/rack/?? or undefined. Each workspace with defined furniture gets a corresponding object facing in the given direction (NOTE: inversions!)

Computers/devices are each assigned an index denoting their location. Precise location of object depends on furniture and direction. Users have defined workspaces.

Unity stub: WorkspaceScript.cs

4 UI Elements

4.1 Computers

Everything below is functionally stubbed in Unity. Artwork/models from existing game. UI's available via right click on computer from existing game. Player can move to different workspace Player can "buy", selects which one and then workspace to place it. Sends buy request to simulation engine. Engine then sends directive to IPCManager to instantiate new computer at location using named xml file. Can recieve directive from IPCManager to remove.

Unity stubs:

- ComputerBehavior (inherits ComponentBehavior)
- ComputerConfigure
- ConfigurationSettings
- ProceduralSettings
- ACLConfigure
- DACAccess
- AssetBehavior

Java UIs

- Email encryption – Email.java; EmailClient.java; EmailData.java
- SSL – in CCSDT: SSLClient.java; SSLData.java; SSLServer.java;
- VPN – TestVPNConnection.java; VerifyCert.java; CertSign.java
- VPN – CCSDT: VPN.java; VPNClient.java; VPNdata.java; VPNpotentials.java; ValidateCert.java

To do:

- software purchase /remove

Purchase of computers (and other items) provides the player with selections from a catelog. The Unity stub is: CatalogBehavior.cs

4.2 Devices

Artwork from existing game. Players move/buy just like computers Unity stubs:

- DeviceBehavior (inherits ComponentBehavior)

Java UIs

- Network filters – in CCSDT: AppFilter.java; Filter.Java; FilterDenialExceptions.java; FilterExceptions.java
- VPN (Same as Computer)
- IDS – IDSDialog.java; IDSData.java

No Java or Unity:

- Link encryptors – see existing game screens: very simple. select 1 of 4 keys and toggle encrypted/clear connections

4.3 Zones

These are areas of the x/y grid. Each zone has different procedural and physical settings. The player should be able to view a map of the zones, e.g., per the current game. Unity stubs:

- ZoneBehavior
- ZoneConfiguration
- ProceduralSettings (Same as Computer)
- PhysicalSettings

4.4 Users

Reuse existing artwork and animations. There are eight different characters and about a dozen animation sequences: sit; stand; walk; type; pound keyboard; idle (look at nails, stretch..) Guards point gun..

Users walk calculated path to other users, other computers, objects (e.g., water cooler), Zone access point. Paths currently reflected in 2d ascii art files per environment. Walkable paths are defined in ascii art txt files in: `https://gitlab.nps.edu/mfthomps/cyberciege/tree/master/game/exec` ms3office.txt; ms3base.txt; mainoffice.txt; smalloffice.txt; workoffice.txt; homeoffice.txt

Chairs move when users sit down.

Doors open when users approach.

The UI configuration limited to "training".

Unity stubs

- UserBehavior
- UserConfigure

IPCManager calls UserBehavior `UpdateStatus` to set thoughts, happiness, productivity. Thoughts displayed when the user is selected, and bubble thoughts when hovered over. Productivity controls whether typing or pounding keyboard.

The simulation engine has only a few cases in which it cares about user movement:

- A "move user" trigger fires to assign a user a new workspace and optionally more the user there right way. This will be reflected in the XML as a move_user field (TBD).
- A "vist user" trigger fires to cause a user to visit another user. This case also requires the UI to notify the simulation engine when visit occurs, i.e., walking user reaches target user.
- A "visit computer" trigger fires to cause a user to visit a computer. This case also requires feedback.
- Users passing through checkpoints, e.g, AI_CheckChokePt in iconicAI. These are only used within one suite of scenarios. Design for this UI/Simulation interation is TBD.

Otherwise, the character movements are as defined in the iconicAI.cpp file. Broadly, users go to their computers if they have them, and wander if they do not. If the user is not achieving goals (see iconicAI) and/or is unhappy, they pound the keyboard, otherwise they type.

IPCManager will (not done yet) call UserBehavior to set walk destinations. UserBehavior will call IPCManager to report entrance to zones and reaching walk destinations.

IPC Manager will call UserBehavior to control bubble speaks; and smoke.

Character naming and textures defined in
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/InitCharacter.cpp`

Animations defined in
`https://gitlab.nps.edu/mfthomps/cyberciege/blob/master/ccsource/prog/iconicAI.cpp`

4.4.1 IT Staff

IT workers and guards are similar to users, but have no productivity goals. The will have walking destinations and bubble speech and thoughts. ITStaffBehavior.cs

4.5 Network connections

Each computer/device pulldown menu will have options for connecting/disconnetion networks. Additionally, there will be a network topology display via which network connections can be modified. See the Network tab in the current game. Constraints on static computers and networks.

Unity stub: NetworkBehavior.cs

4.5.1 Network discovery

Java UI: Discover.java; DiscoverData.java

4.6 Attack log

AttackLog.java Running log of text. IPCManager calls to add. Button color changes on new entries. Includes a scenario-dependent varient: CasaLog.java

4.7 Wireshark knockoff

Java UIs: CCPacket, CCPacketDetails; CCPacketReader; CyberChark

4.8 Triggers

- Objectives/Phases: Unity stub: ObjectivesBehavior
- Camera movements: to user/computer/device/predefined positions.
- Ticker messages (scrollers) and withdrawl of messages. Unity stub: ScrollingText.cs
- Tool tips. Unity stub: ToolTipScript.cs
- Multiple choice questions – ABC.java
- Checklists – CheckList.java
- Yes/no – YesNo.java
- Fire / Smoke

4.9 Scenario Player

Unity stub: Assets/PlayerScripts/PlayerScript.cs For selecting "campaign" and scenario. Track which scenarios completed. Manage prerequisites. View logs. Java at: https://gitlab.nps.edu/mfthomps/cyberciege/tree/master/Ccse/SAT/Cplayer

4.10 Encyclopedia

Uses MS HTML workshop help. Dozens of pages. Triggers select default page if help is invoked.

4.11 Msc

Selecting users/computers/devices results in stargate rings

User clearance and background check dialogs (not per user).

Briefings

Help tips

Animated posters on walls.

Scenario replay controls (probably use existing Java).

View game log?

Game status with GameStatusBehavior.cs