**Vi-Char**

**Motion Capture Division**

**Design Specifications**

Daniel Porter

Ian Earle

Travis Dos Santos-Tam

Max Gobel

**Introduction**

This document describes the Motion Capture design specifications for the larger Vi-Char project, and details both the overall thought process behind the design decisions as well as the specific ways in which this project will interact with other components of the project at large.

The Vi-Char system is incorporating a number of different technologies for a wide variety of purposes. Within this system, motion capture is being used as a method of input, and so the system is to be designed in a way as to best facilitate a smooth and easy to understand user experience. The motion capture tools act as one of the primary ways which the users interact with the system at large, and it is our goal as a group to design a product which serves as a seamless interface to the user while efficiently providing all of the required information to the Game Engine.

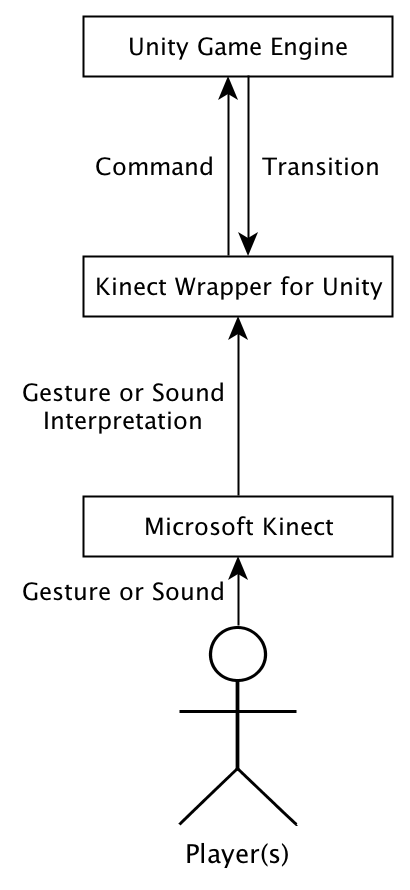
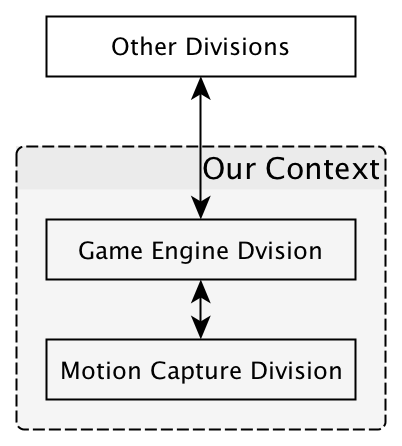
**Architecture Design**

This document describes a software architecture for the motion — and by extension, sound — capture component needed to support the overall ViChar game. The goal of the architecture is to describe all of the software components, their responsibilities and behavior and their interfaces.

There are two major subcomponents to our group’s offerings. The first is the Microsoft Kinect, which captures user’s physical actions and voice commands. Motions that the players may gesture include jumping, rolling, basic directional movements and primary and secondary weapon attacks. Interfaces our Kinect module will offer include gesture and sound interpretation, which essentially means that the physical movements it sees and the sounds it hears are digitized and those “messages” are passed onto other modules.

The Kinect wrapper for Unity will allow our group to integrate Kinect easily into Unity. Using the interpreted messages from Kinect, the wrapper will match gestures and sounds to appropriate commands to send to the Unity game engine. For example, a jump gesture perceived by the Kinect will be passed to the wrapper which will issue a jump command to the game engine.

Finally, the game engine will receive our messages to effect its state — character position, score, etc. Additionally, the game engine will be able to pass messages back to the Kinect wrapper. The messages will tell the Kinect wrapper what kind of motions and sounds are expected, especially between stages of the game, etc.



**Module Design**

**Kinect Hardware**

The Kinect is a device which contains four microphones, one RGB camera, and one IR depth camera. These sensors are all used to accurately model the world it can see and recognize both specific humans and their motions.

The purpose of the Kinect hardware in this project is to gather input from the surrounding world for another module to interpret. The Kinect will gather sound, video, and depth information from the surrounding environment.

The Kinect hardware provides three data streams, one for sound, one for video, and one for depth information. This information will be sent to a computer via a USB cable.

The Kinect requires a computer with all appropriate Kinect drivers to be installed. This computer must be actively listening for Kinect data.

**Motion Interpreter**

The motion interpreter receives raw data from the Kinect and look for specific gestures.

The purpose of this module is to use the Kinect data to model the physical world. This information can then be analyzed to recognize specific motions and sounds which will be sent to another module.

The motion interpreter provides a listener for the Kinect hardware data streams, as well as a way to communicate with the Unity engine to listen for a change in requested information. If the game engine is requesting a high score to be input, the Kinect will change the set of gestures it is looking for to recognize a player entering their high score.

The motion interpreter requires a working connection to a Microsoft Kinect and a connection to the game engine.

**Unity3D Module**

The Unity3D module receives data from the motion interpreter and applies it to the game world.

The purpose of the Unity3D module is to let the rest of the game engine know that a specific gesture has been completed.

The Unity3D module provides a hook for the motion interpreter to communicate with. It also sends messages to the game engine when needed.

The Unity3D module requires the rest of the game engine to receive the created messages.

**Other Design Views**

The conceptual design of our system is fairly closed, meaning that it interacts very sparsely with the other components of the Vi-Char system at large. The main connection is one from our project to the functioning game engine, a relay system which will consist primarily of us passing “events” to the game engine as they are triggered by player input, allowing the game (and by extension, the rest of the system) to respond accordingly.

The integration will occur along this same single thread. As we have no need to pass information directly to the most of the system, the only connection will be directly from our project to the game engine. As this system is will focus almost entirely on us passing information to the game engine, though there are one or two cases in which the reverse will occur, our system will primarily be modeled off of event-driven architecture. We have a finite list of actions which the user can do, and each of these will be tied to a unique event that can be passed on to the game engine. This is the method through which our system will interact with the Game Engine, and thus, the Vi-Char system at large.

**Design Rationale**

The design choices laid out in this document were primarily motivated by an object oriented paradigm. By designing our software in the way we did, each section has a clear purpose and can function mostly independently from the other pieces, allowing a smooth development process.

**Implementation Notes**

The Motion Capture portion of the project will be implemented as a module to the game engine, communicating with it directly through either message passing, or calls from the game engine to a library.

Messages will likely be passed on a frame-by-frame basis, to facilitate better cohesion with the game engine. Additionally, continuous actions (such as turning) will either have a start and stop event, or will be sent with magnitudes until the action is completed.

Player role will be dictated by player number (as determined by the Kinect). Players will be able to switch roles if desired.

**Glossary & References**

**References:**

Microsoft Kinect for Windows SDK (<http://www.microsoft.com/en-us/kinectforwindows/)>

Kinect Dev Tutorials (<http://www.kinecthacks.com/kinect-sdk/>)

Develop Kinect Community (<http://developkinect.com/>)