Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>1</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>1.1 What is Cambot?</td>
<td>4</td>
</tr>
<tr>
<td>1.2 What is Machinima Sequencing Tool (MST)?</td>
<td>5</td>
</tr>
<tr>
<td>2. INSTALLATION</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Install the Virtual Backlot</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Install Cambot/MST</td>
<td>7</td>
</tr>
<tr>
<td>3. HOW TO USE CAMBOT</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Running Cambot</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Input file specifications</td>
<td>8</td>
</tr>
<tr>
<td>3.2.1 Scene file</td>
<td>8</td>
</tr>
<tr>
<td>3.2.2 World files</td>
<td>12</td>
</tr>
<tr>
<td>3.2.3 Director files</td>
<td>13</td>
</tr>
<tr>
<td>4. HOW TO USE MACHINIMA SEQUENCING TOOL</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Running Machinima Sequencing Tool</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Tutorial</td>
<td>13</td>
</tr>
<tr>
<td>4.3Hints</td>
<td>17</td>
</tr>
<tr>
<td>5.1 Cambot</td>
<td>18</td>
</tr>
<tr>
<td>5.2 Machinima Sequencing Tool</td>
<td>18</td>
</tr>
<tr>
<td>6. MST CHEAT SHEET</td>
<td>19</td>
</tr>
<tr>
<td>6.1 Shots and Blockings</td>
<td>19</td>
</tr>
<tr>
<td>6.2 Character Animations</td>
<td>31</td>
</tr>
<tr>
<td>7. EXTENDING THE UT 2004 VIRTUAL BACKLOT</td>
<td>32</td>
</tr>
<tr>
<td>7.1 Making New Places</td>
<td>32</td>
</tr>
<tr>
<td>7.2 Making New Characters</td>
<td>32</td>
</tr>
<tr>
<td>7.2.1 Adding New Characters Via Skinning</td>
<td>32</td>
</tr>
<tr>
<td>7.2.2 Adding New Characters With New Model</td>
<td>33</td>
</tr>
<tr>
<td>7.2.3 Adding New Characters With New Model and Animations</td>
<td>33</td>
</tr>
<tr>
<td>8. EXTENDING CAMBOT</td>
<td>34</td>
</tr>
<tr>
<td>9. EXTENDING MST</td>
<td>34</td>
</tr>
<tr>
<td>10. WRITING YOUR OWN VIRTUAL STUDIO BACKLOT</td>
<td>34</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>39</td>
</tr>
</tbody>
</table>
1. Introduction

Machinima (http://en.wikipedia.org/wiki/Machinima) is a low-cost alternative to full production filmmaking that repurposes computer game engines. However, creating quality cinematic visualizations with existing machinima techniques still requires a high degree of talent and effort. We introduce a pair of technologies designed to reduce the cost and time of developing machinima movies.

The technologies described in this document are designed to facilitate rapid prototyping, production, and editing of basic dialogue-based films. With these technologies, you can script a film and shoot it quickly and easily. If you need to make changes, the script can easily be modified and the entire film re-shot in a short amount of time.

Cambot is an intelligent agent that “shoots a script.” In essence, Cambot is a “synthetic movie director,” which makes all the decisions that a real director might make in full production filmmaking. Cambot does location scouting in a virtual world, determines how to arrange the actors in each scene, determines how to shoot each scene, and coordinates the actors and cinematography. Suppose you want to make a movie, but don’t know how to block actors1, set up shots, or coordinate actors and cinematography. If you can write a movie script – dialogue and accompanying scene descriptions – Cambot can shoot your movie.

Figure 1. An example fragment of a script.

1 Blocking is a term referring to the positioning of actors in a scene.
Figure 2. Shots produced by Cambot, corresponding to Figure 1.

Machinima Sequencing Tool (MST) is a machinima authoring tool designed to allow rapid prototyping of machinima sequences. MST allows the user to engage in a cycle of creation where he or she can quickly specify the scenes, dialogue, and shots that he or she wishes to see in a movie, view the results, and return to editing. Unlike Cambot, the MST user interface provides the creator full control of choice of blockings and shots.

Both Cambot and MST are designed to work with a virtual world. The example virtual world that we provide is a mod to Unreal Tournament 2004. However, any virtual world that read specialized instructions from Cambot/MST over a TCP/IP connection. Our UT2004 virtual world implements what we call a “virtual backlot”: a set of disparate sets representing different places action can take place. For demonstration purposes, our virtual backlot currently only contains a city street and a portion of the Georgia Tech campus.
1.1 What is Cambot?
Cambot takes a script as input and produces a cinematic visualization. Unlike other virtual cinematography systems, Cambot favors an offline algorithm coupled with an extensible library of specific modular and reusable facets of cinematic virtual cinematography is a tight coordination between the positions and movements of the camera and the actors. More information about how Cambot works can be found in the publication: A Lightweight Intelligent Virtual Cinematography System for Machinima Production by Elson and Riedl (Proceedings of the 3rd Annual Conference on Artificial Intelligence and Interactive Digital Entertainment, 2007). The following information is an overview of the information available in the paper.

Filmmaking begins with a script. Figure 1 shows a script in a format similar to that used in the film industry. The core elements of the script are the dialogue and directives on how the scene should look and sound. The primary input into the Cambot system is an XML file that duplicates many of the same elements found in an actual script, including dialogue and constraints about scene location, actor blockings, and viewpoint. Note that Cambot does not perform natural language understanding and thus does not comprehend the content of any dialogue.

Cambot must shoot the script, satisfying as many of the constraints as possible. To accomplish this, Cambot uses a combination of search and dynamic program to identify shots and blockings that best fit each beat in each scene. A library of canonical blockings and shots is encoded into Cambot. Each beat must be “covered” by a blocking and a shot. Actors must be told where to stand and the shot conveys the scene to the watcher.
Figure 3.a. Schematics of character blockings. The first shows two characters standing facing each other. The second shows one character approaching the other.

Figure 3.b. Schematics of two types of shots. The first is an “apex” shot that establishes characters in a scene. The second is a panning shot where the camera sweeps across the scene.

Cambot additionally searches for the best location in the virtual backlot world to set each scene. A “stage” is a piece of virtual real estate, typically a rectangle, which is placed such that there are no possible occlusions inside the real estate. This ensures that we do not have to detect or plan around unwanted occlusions. Once the stage location is selected, blockings and shots are selected.

Finally, you may wish to control the aesthetic style of Cambot. Cambot comes with a built in heuristic that favors moving shots and avoiding cuts. A new director style heuristics can be provided as an input file.

1.2 What is Machinima Sequencing Tool (MST)?

The Machinima Sequencing Tool (MST) is built on top of Cambot. MST enables the user to make all decisions that Cambot would make. That is, MST allows a user to rapidly prototype a machinima movie by specifying scene locations, dialogue and other actions, blockings, and shots. MST supports an interactive editing cycle.

---

2 Cambot supports different shaped stages. We have implemented two stage shapes, a rectangle and an elongated rectangle, which should be sufficient for most action.
Figure 4. MST editing screens.
2. Installation
There are two distinct systems that you must set up. The first is Cambot/MST. These systems are implemented in Java and compiled into a stand-alone JAR file. The second is the virtual backlot. We provide a mod to Unreal Tournament 2004 that can accept instructions from Cambot/ and a virtual backlot map that has the geometry and avatars.

2.1 Install the virtual backlot
For unimportant legacy reasons, the UT2004 mod is called “ALERT.” To install the ALERT mod:

1. Install Unreal Tournament 2004.
2. Patch UT2004 to the latest patch (v. 3369):
   \texttt{ut2004-winpatch3369.exe}
3. Copy cambot\ALERT to c:\ut2004\ALERT
4. Copy cambot\animations\* to c:\UT2004\animations\*
5. Copy cambot\textures\* to c:\UT2004\textures\*
6. Copy cambot\staticmeshes\* to c:\UT2004\staticmeshes\*
7. Copy cambot\system\* to c:\UT2004\system\*

2.2 Install Cambot/MST
No installation required. You should be able to run Cambot and MST from the batch files provided.

3. How to use Cambot
Cambot takes as input a script that has been encoded in XML. The structure of a Script is as follows: A Script object is a list of Scenes. Each Scene contains one or more Beats. Finally, Beats are made up of one or more Actions – something that actually “goes on” in the script such as dialogue or a gesture.

Scenes and Beats may be annotated with one of several types of constraints. These constraints allow users to shape the aesthetics of the script. There are four types of constraints:

- **BlockingConstraint.** This type of constraint is analogous to writing “Jim stands next to Joe” or “Jim walks away from Joe” in a natural-language script. It allows script writers to impose restrictions on where the characters in a scene stand in relation to one another.
- **LocationConstraint.** This type of constraint is analogous to writing “INT. OFFICE,” or “EXT. STREET” in a natural-language script. It allows script writers to impose restrictions on where actions of a scene may be located.
- **ViewConstraint.** This type of constraint is analogous to writing “CLOSE on Jim” in a natural-language script. It allows script writers to impose restrictions on the types
of shots that may be used to “cover” a portion of the script.

- SceneConstraint. This type of constraint is analogous to a movie director’s aesthetic sensibilities. Its implementing classes allow script writers to impose restrictions on the overall look and feel of the scene, e.g., how often the camera should cut between different shots.

### 3.1 Running Cambot

1. Run the UT2004 virtual backlot:
   ```
   alert.bat
   ```

2. Run Cambot:
   ```
   cambot.bat localhost script3.xml world0.xml director0.xml
   ```
   XMLDemo is the main class. The first argument is the location of the virtual backlot. The second argument is the script file. The third argument is the world file. The fourth argument is the optional director file. Cambot will use the built-in heuristic if the director file is left off.

### 3.2 Input file specifications

The following describes how to create your scene file, world file, and optional director file.

#### 3.2.1 Scene file

The following takes you through an example in which we set up a simple scene in an XML file. Our example begins with establishing a list of scenes (although there will only be one scene). A scene needs to know what characters will participate in it.

```xml
<Scenes>
  <!-- Scene 1 -->
  <Scene DefaultViewConstraintWeight="100" DefaultSceneConstraintWeight="100">
    <SceneCharacters>
      <Character Name="Sarah" Species="HUMAN" Gender="MALE" Height="MEDIUM" Age="86"/>
      <Character Name="Jeeves" Species="HUMAN" Gender="MALE" Height="MEDIUM" Age="86"/>
    </SceneCharacters>
  </Scene>
</Scenes>
```

In theory, characters can have different species, genders, and heights. However, Cambot does not support this, so all character avatars must be of the same _eye-height_ – the distance from the ground to their eyes. The virtual world (e.g., UT2004) must be able to resolve the names into avatars.

Scenes have one or more beats. Beats are made up of actions and constraints. A beat must specify which characters are in the beat (a subset of the characters in the scene). We add the first beat below.

```xml
<Beats>
  <!-- beat 0 -->
  <Beat>
    <BeatCharacters>
      <CharacterRef>Jeeves</CharacterRef>
      <CharacterRef>Sarah</CharacterRef>
    </BeatCharacters>
    <RequiredBlockingConstraints>
```

<RequiredBlockingConstraint
BlockingConstraintType="BlockingRelationshipConstraint"
DesiredPresence="true">
<Character1>Sarah</Character1>
<Character2>Jeeves</Character2>
<BlockingRelationType>APPROACHING</BlockingRelationType>
</RequiredBlockingConstraint>
</RequiredBlockingConstraints>
<WeightedViewConstraints>
<WeightedViewConstraint ViewConstraintType="ViewAnnotationConstraint"
Weight="50">
<Character>Jeeves</Character>
<DesiredPresence>true</DesiredPresence>
<ViewAnnotationFeature>INTRODUCE</ViewAnnotationFeature>
</WeightedViewConstraint>
</WeightedViewConstraints>
<Actions>
<Action ActionType="DialogueAction" Delay="3" Duration="3">
<Character>Sarah</Character>
<Dialogue>Soon, there will be no turning back.</Dialogue>
</Action>
<Action ActionType="GestureAction" Delay="3.5" Duration="1">
<Character>Jeeves</Character>
<Gesture>LookLeft</Gesture>
</Action>
</Actions>
</Beat>
</Beats>

Breaking this down, we have a beat, specifying references to the characters that belong. The beat establishes one required constraint and one weighted constraint. A required constraint must occur, meaning that any selection of blockings or shots must meet the specification. In this case the required constraint is on the way the beat is blocked: Sarah must approach Jeeves. A weighted constraint is preferred if possible, but can be relaxed otherwise. Weighted constraints have a weight indicating how rewarding it will be to Cambot to satisfy the constraint. In this case, the weighted constraint is on the visualization of the beat: Jeeves should be introduced, meaning his location in the set is established, usually by a medium or long shot.

The first beat has two actions. The first action is a dialogue action, specifying that Sarah must speak the following line of text. Actions have durations and delays. The duration specifies how long the action will last. The delay specifies how many seconds from the start of the beat the action should begin. Note that this allows parallel overlapping actions, as is the case here. The second action is a gesture action. A gesture is actually any preset animation that an avatar can perform in the virtual world.

Suppose that is the last beat. Now, we may want to provide constraints on the overall scene. For example, we might want to specify some constraints on the location of the scene. First a bit more detail on the virtual backlot. Cambot understands the virtual backlot to be made up of boxes that specify “shootable” spaces. We call these “Open Boxes.” Each open box can be tagged with descriptive labels. Additionally, there can be specific points in the world that are named. Before we close out our scene, we add some information to help Cambot select the location of the scene.
In this example, there are two required constraints on the location. The first is that the scene is shot in an open box that is labeled as “STREET”. The second is that the scene occurs within 300 units of location called “Intersection.”

Finally, our scene is closed, and that is our only scene.

The full scheme for a script file is as follows:

```
Scenes := [Scene+]
Scene := [SceneCharacters] [Beats] [RequiredLocationConstraints?]
   [WeightedLocationConstraints?]
SceneCharacters := [Character+]
Beats := [Beat+]
Beat := [BeatCharacters]
   [RequiredBlockingConstraints?] [WeightedBlockingConstraints?]
   [RequiredViewConstraints?] [WeightedViewConstraints?]
   [Actions]
Actions := [Action+]
Action := [Character?] [Dialogue?] [Gesture?] [Direction?]
BeatCharacters := [CharacterRef+]
RequiredLocationConstraints := [RequiredLocationConstraint+]
WeightedLocationConstraints := [WeightedLocationConstraint+]
RequiredBlockingConstraints := [RequiredBlockingConstraint+]
WeightedBlockingConstraints := [WeightedBlockingConstraint+]
RequiredViewConstraints := [RequiredViewConstraint+]
WeightedViewConstraints := [WeightedViewConstraint+]
RequiredLocationConstraint := [DesiredPresence?] [OpenBoxLabel?] [PointName?]
   [Threshold?] [ShouldBeWithThreshold?]
WeightedLocationConstraint := [DesiredPresence?] [OpenBoxLabel?] [PointName?]
   [Threshold?] [ShouldBeWithThreshold?]
RequiredBlockingConstraint := [Character1] [Character2?]
   [BlockingRelationType] [BlockingLabel]
WeightedBlockingConstraint := [Character1] [Character2?]
   [BlockingRelationType] [BlockingLabel]
RequiredViewConstraint := [Character] [DesiredPresence?] [DesiredComparison?]
   [ShotType?] [ViewAnnotationFeature?]
   [ViewAnnotationType?]
WeightedViewConstraint := [Character] [DesiredPresence?] [DesiredComparison?]
   [ShotType?] [ViewAnnotationFeature?]
   [ViewAnnotationType?]
CharacterRef := String
Character1 := String
Character2 := String
Dialogue := String
Gesture := String
```
Direction := Boolean
DesiredPresence := Boolean
DesiredComparison := -1 | 0 | 1
OpenBoxLabel := String
PointName := String
Threshold := Double
ShouldBeWithThreshold := Boolean
BlockingRelationType := “APPROACHING” | “TALKING_TO” | “FACING” | “NEARBY”
BlockingLabel := “ANIMATED” | “STATIC”
ShotType := “CU” | “ECU” | “ELS” | “LS” | “LSTOMCU” | “MCU” | “MCUTOCU” | “MCUTOMS” | “MS” | “MSTOMCU”
ViewAnnotationFeature := “INTRODUCE” | “SEE_POINT_OF_GAZE” | “INCREASING_INTENSITY” | “DECREASING_INTENSITY”

<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute</th>
<th>Type</th>
<th>Req?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene</td>
<td>DefaultViewConstraint</td>
<td>Integer</td>
<td>Yes</td>
<td>Weight scale for view constraints. That is, all constraint weights should add up to this.</td>
</tr>
<tr>
<td></td>
<td>Weight</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DefaultSceneConstraint</td>
<td>Integer</td>
<td>Yes</td>
<td>Weight scale for scene constraints. That is, all constraint weights should add up to this.</td>
</tr>
<tr>
<td></td>
<td>Weight</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beat</td>
<td>SetInheritsSceneView</td>
<td>Boolean</td>
<td></td>
<td>Should this beat inherit the constraints of the scene?</td>
</tr>
<tr>
<td></td>
<td>Constraint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Name</td>
<td>String</td>
<td>Yes</td>
<td>Name of the character. Must map to an avatar in the virtual backlot.</td>
</tr>
<tr>
<td></td>
<td>Species</td>
<td>String</td>
<td></td>
<td>Not implemented.</td>
</tr>
<tr>
<td></td>
<td>Gender</td>
<td>String</td>
<td></td>
<td>Not implemented.</td>
</tr>
<tr>
<td></td>
<td>Height</td>
<td>String</td>
<td></td>
<td>Not implemented.</td>
</tr>
<tr>
<td></td>
<td>Age</td>
<td>Integer</td>
<td></td>
<td>Not implemented.</td>
</tr>
<tr>
<td>RequiredLocationConstraint</td>
<td>LocationConstraintType</td>
<td>String</td>
<td>Yes</td>
<td>Implemented values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• OpenBoxLabelConstraint: use OpenBoxLabel tag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• StageProximityConstraint: use PointName, Threshold, and ShouldBeWithThreshold tags.</td>
</tr>
<tr>
<td></td>
<td>DesiredPresence</td>
<td>Boolean</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>WeightedLocationConstraint</td>
<td>LocationConstraintType</td>
<td>String</td>
<td>Yes</td>
<td>Same as above.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RequiredBlockingConstraint</td>
<td>BlockingConstraintType</td>
<td>String</td>
<td>Yes</td>
<td>Implemented values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• BlockingRelationshipConstraint: use BlockingRelationType tag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• BlockingLabelConstraint: use BlockingLabel tag</td>
</tr>
<tr>
<td></td>
<td>DesiredPresence</td>
<td>Boolean</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>WeightedBlockingConstraint</td>
<td>BlockingConstraintType</td>
<td>String</td>
<td>Yes</td>
<td>Same as above.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RequiredViewConstraint</td>
<td>ViewConstraintType</td>
<td>String</td>
<td>Yes</td>
<td>Implemented values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• ViewAnnotationConstraint: use ViewAnnotationFeature tag</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• FaceSeenInConstraint: use ShotType and DesiredComparison tags</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• ViewSpeakersConstraint: use ShotType tag</td>
</tr>
<tr>
<td></td>
<td>DesiredPresence</td>
<td>Boolean</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>WeightedViewConstraint</td>
<td>ViewConstraintType</td>
<td>String</td>
<td>Yes</td>
<td>Same as above.</td>
</tr>
</tbody>
</table>
3.2.2 World files
Cambot maintains a 2D map of the virtual backlot, provided by the world file. The world file also tells Cambot which shots can be used. From Cambot’s perspective, the world is made up of Open Boxes. Each Open Box should be completely occlusion free, meaning that no matter where an object and a camera is placed in the box, the camera should be able to see the object. Boxes are tagged so that Cambot can make some simple decisions about where to stage scenes. All Open Boxes are assumed to be on the same plane – the virtual backlot needs to meet this requirement. Note that contiguous boxes should be slightly overlapping, otherwise Cambot will treat them as non-contiguous.

The following sets up a single box with a single tag.

```xml
<World>
  <Boxes>
    <Box>
      <Corner1 x="-29824" y="-6848"/>
      <Corner2 x="-27840" y="-7168"/>
      <height>100</height>
      <BoxFeatureSet>
        <BoxFeature>STREET</BoxFeature>
      </BoxFeatureSet>
    </Box>
  </Boxes>
</World>
```

The world file can also specify specific points of interest. The following indicates an “intersection” in our “street”.

```xml
<PointsOfInterest>
  <PointOfInterest xLoc="-28600" yLoc="-7000" Height="100" Name="Intersection"/>
</PointsOfInterest>
```

Finally, we want to specify which shots can be used.

```xml
<Shots>
  <Shot>Conv1A</Shot>
  <Shot>Conv1B</Shot>
  <Shot>Conv1C</Shot>
</Shots>
```

The full scheme for a world file is as follows:

```
World := [Boxes] [PointsOfInterest?] [Shots]
Boxes := [Box+]
PointsOfInterest := [PointOfInterest+]
```
Shots := [Shot+]
Box := [Corner1] [Corner2] [height] [BoxFeatureSet]
Corner1 := Point2D
Corner2 := Point2D
height := float
BoxFeatureSet := [BoxFeature+]
BoxFeature := String
Shot := String

<table>
<thead>
<tr>
<th>Tag</th>
<th>Attribute</th>
<th>Type</th>
<th>Req?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point2D</td>
<td>x</td>
<td>Double</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>y</td>
<td>Double</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>PointOfInterest</td>
<td>xLoc</td>
<td>Double</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>yLoc</td>
<td>Double</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td>String</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Height</td>
<td>Integer</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

3.2.3 Director files
Cambot comes with a default directorial heuristic, meaning that if you do not specify a director file, Cambot will prefer shots that are moving and minimize cuts. Providing a director file allows you to override these defaults. See the provided director file for an example of a directorial heuristic that prefers static shots and lots of cuts.

4. How to use Machinima Sequencing Tool
MST uses a wizard-like user interface to allow the user to specify what happens in the movie and how it looks.

4.1 Running Machinima Sequencing Tool
1. Run the UT2004 virtual backlot:
   alert.bat

2. Run MST:
   MST.bat

4.2 Tutorial
The interface should be fairly self-explanatory. This section walks through the creation of a simple machinima movie.

The first screen asks you to name your movie and to select a world file (XML).

1.1. Use the Find button to select the world file that you desire. Note that the world file must be in the same directory as the JAR file. The world file provided is world0.xml.

1.2. Press “OK” to start your movie.

The next screen allows you to specify which characters will be in your movie. You must specify all characters that will be used in the movie, even if they are not in every scene or beat. A drop down box lists the characters provided. If you have added new characters to the virtual backlot, you may also specify them with “New Character”.
2.1. Add Sarah by selecting her name from the pull-down menu and pressing “Add Character”.

2.2. Add Jeeves by selecting his name from the pull-down menu and pressing “Add Character”.

2.3. Press “Create Scene”.

The third screen asks you to specify your first scene. Recall that a movie is made up of scenes. Scenes are made up of beats. And Beats are made up of actions. When a new scene is created, it is associated with a location.

3.1. Give your scene a name: “Servant and master”.

Select which characters will be in the scene. You must specify all characters who will be in any subsequent beat, even if they are not in every single beat.

3.2. Check the boxes next to Sarah and Jeeves.

Select where the scene should be located. This is the center point of an imaginary box in which characters and camera will be eventually placed.

3.3. Click on a point in the map near the point labeled “intersection”.

MST does not check whether this imaginary box extends outside the Open Boxes, so if you click too close to a wall, you can get occlusions or characters stuck inside walls. You can rotate the imaginary box in which the scene occurs.

3.4. Set the angle to “90”.

3.5. Click “Add First Beat”.

The beat creation scene allows you to create a single beat. Recall that a beat is a short piece of the movie, usually just a moment in time consisting of a piece of dialogue or two. The beat happens at the scene location.

4.1. Give you beat a name: “Greetings”.

Select which characters will be in your beat.

4.2. Check the boxes next to Sarah and Jeeves.

Choose how you want to block your characters and how you want to shoot the beat. See Section 6 for meaningful descriptions of the blockings and shots.

4.3. Select “Conv3” from the blocking pull-down menu.

4.4. Select “Conv3D” from the shot pull-down menu.

Beats need actions. Let’s have Jeeves say something.

4.5. Use the Action pull-down menu to select “Dialogue”. You will see some additional options appear.

4.6. Select Jeeves from the pull-down menu that appeared just below.

4.7. Enter some text in the text field: “You rang?”
The delay is from the start of the beat. This allows you to have multiple actions happening simultaneously or overlapping.

4.8. Set the delay to 1 second.
4.9. Set the duration to 5 seconds.
4.10. Press “Add Action”.

The action list will be updated so you know what actions are in the beat. Now, let’s have Sarah respond.

4.11. Select “Dialogue” from the action pull-down menu.
4.12. Select Sarah from the pull-down menu that appeared just below.
4.13. Enter text into the text box: “It’s about time you got here.”
4.15. Set the duration to 5.
4.16. Press “Add Action”.

Let’s add a second beat to this scene.

4.17. Press “Add Next Beat”.

The screen looks the same but is in fact an empty second beat.

5.1. Give your beat a name: “Jeeves compliant”.
5.2. Check the boxes next to Sarah and Jeeves.
5.3. Select “Conv1” from the blocking pull-down menu.
5.4. Select “Conv1B” from the shot pull-down menu.
5.5. Select “Dialogue” from the action pull-down menu.
5.6. Select Jeeves from the pull-down menu that appeared just below.
5.7. Add a line of dialogue to the text box: “How may I be of assistance?”
5.8. Set the delay to 0.
5.9. Set the duration to 5.
5.10. Press “Add Action”.
5.11. Press “Done Scene and Movie”.

You are now on the movie review screen. You can see the outline of your movie on the left. Clicking on any element in the outline gives details on the right. From here you can edit any of the scenes or beats or play the movie. Let’s play your movie.

6.1. At the bottom of the screen, press the “Play Movie” button.

You should see your beats play out Unreal Tournament.
Once your movie is done, control returns to the MST screen. Let’s make some changes to your movie. We will change the location of your scene and make the scene longer by adding more beats.

7.1. Highlight your scene (“Servant and master”) on the left-hand side.
7.2. Press “Edit”.

You should see the scene set up screen, but this time the information about your scene is there for you to view. Let’s change the location of your scene.

8.1. On the map, scroll down until you see a black box.
8.2. Click somewhere in the big black box.

You have just relocated your movie to Georgia Tech campus. But before we watch your new movie, let’s add a third beat. Note the new button on the screen called “Add Beat to End”.

8.3. Press the “Add Beat to End” button.

You should see the beat creation screen.

9.1. Give your beat a name: “A request”.
9.2. Check the boxes next to Sarah and Jeeves.
9.3. Select “Conv1” from the blocking pull-down menu.
9.4. Select “Conv1C” from the shot pull-down menu.
9.5. Select “Dialogue” from the action pull-down menu.
9.6. Select Sarah from the character pull-down menu that appeared just below.
9.7. Add some dialogue to the text box: “Bring me wine.”
9.8. Set the duration to 5.
9.9. Press “Add Action”.
9.10. Press “Done adding beat”.

You are back on the movie review screen.

10.1. Press “Play Movie” to see your new, longer movie in the new location.
10.2. Use screen capture software (we recommend Camtasia Studio) to record your movie.
10.3. Upload it to YouTube.
10.4. Send it to all your friends.

You can see the movie that is produced from this tutorial in the movies folder. You can see a video walkthrough of this tutorial in the movies folder.
4.3 Hints

• Next to the blocking choice pull-down menu is a button labeled “b”. Pressing this button allows you to select how you want characters assigned to the different positions in a blocking (See Section 6).
5. Known issues

5.1 Cambot

- If characters heads seem to be cut off, it is probably because the characters are not scaled correctly for the camera shots. Best way to do this is to use MST to create a close-up shot of a character. The character should appear on the screen approximately like this:

- If UT2004 is left open long enough, it will cycle to a random map. You must restart UT2004 when this happens.

5.2 Machinima Sequencing Tool

- You may see a character raise or lower after walking. This has something to do with the bounding boxes around avatars. This can be adjusted in UnrealEd by opening the animations screen and fiddling with the bounding boxes around the animation in question.

- Sometimes the characters fail to walk. This happens when the avatar is stuck on some piece of geometry. Moving the scene location a little bit might help.

- Avatars remain where they are last positioned. When you play the movie a second time, you may find characters remaining where they were and getting in the way. I recommend ending all movies with a “reset” scene in which characters are move to some unused location.

- The world file must be in the same working directory as the cambot.jar file.

- Sometimes characters fail to stop walking. Avatars check every “tick” whether they are within a prescribed radius of their final destination point. We don’t know exactly why this happens sometimes, but it seems to be dependent on the computer (and possibly processor speed). We surmise that there is a race-condition somewhere deep in UT2004.
6. MST cheat sheet

6.1 Shots and blockings
Blockings and shots refer to the characters as HMA_1, HMA_2, etc. In Cambot and MST, character names are bound to these variables. Blockings are outlined. Shots that work with the blocking are listed beneath.

<table>
<thead>
<tr>
<th>Conv1</th>
<th>A blocking with two human male adults facing one another at a few arm’s lengths (i.e., not too friendly, but not too hostile).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Conv1A</strong></td>
<td>An over-the-shoulder close-up of HMA_1.</td>
</tr>
<tr>
<td><strong>Conv1B</strong></td>
<td>A medium wide shot of both from the perpendicular.</td>
</tr>
</tbody>
</table>
Conv1C  An over-the-shoulder close-up of HMA_2.

Conv1D  An establishing shot along the perpendicular axis that pushes in and tilts down from the sky.

Conv1E  A trick shot that dramatically transitions from an over-the-shoulder close-up of HMA_1 to an over-the-shoulder close-up of HMA_2 by spinning the camera around and moving it to the other shoulder.

Conv1F  A trick shot that dramatically transitions from an over-the-shoulder close-up of HMA_2 to an over-the-shoulder close-up of HMA_1 by spinning the camera around and moving it to the other shoulder.
Conv1G  An over-the-shoulder medium-close-up of HMA_1.

Conv1H  An over-the-shoulder medium-close-up of HMA_2.

Conv1I  Variable-length shot that pushes from medium-close-up over-the-shoulder to close-up over-the-shoulder of HMA_1.

Conv1J  Variable-length shot that pushes from medium-close-up over-the-shoulder to close-up over-the-shoulder of HMA_2.
Conv1K  Static shot of both from a wide, low, oblique angle.

Conv1L  Static shot of both from a very wide, high, oblique angle.

Conv1M  Starts out like Conv1K (a low, wide shot of both from an oblique angle) but trucks over laterally to the mirror image.

Conv1N  An over-the-shoulder medium-shot of HMA_1.
Conv1O  An over-the-shoulder medium-shot of HMA_2.

Conv1P  Variable-length shot that pushes from over-the-shoulder medium-shot to over-the-shoulder medium-close-up of HMA_1.

Conv1Q  Variable-length shot that pushes from over-the-shoulder medium-shot to over-the-shoulder medium-close-up of HMA_2.
Conv2

A blocking with two human male adults standing side by side looking at something along the axis perpendicular to the one on which they stand.

Conv2A

A medium wide shot of both from the perpendicular.

Conv2B

An oblique shot of both characters, almost an over-the-shoulder shot of HMA_1 over HMA_2's shoulder.

Conv2C

An oblique shot of both characters, almost an over-the-shoulder shot of HMA_2 over HMA_1's shoulder.
Conv2D  A medium wide over both their shoulders toward what they are looking at.

Conv2E  A medium close up shot of both from the perpendicular.

Conv2F  A variable-length shot that goes from a medium to medium-close-up of both from the perpendicular.

Conv2G  A variable-length shot that goes from an medium-close-up to a medium of both from the perpendicular.
Conv2H  Moving shot that essentially goes from Conv2B to Conv2E.

Conv2I  Moving shot that essentially goes from Conv2E to Conv2C.

Conv3  A blocking with one human male adult arriving on the stage to address a second human male adult. This blocking ends with the characters in position for Conv1 blocking.

Conv3A  A static extra-long-shot establishing shot of both.
**Conv3B**  A static establishing shot of both from a loose over-the-shoulder of HMA_1 over HMA_2.

**Conv3C**  A shot similar to Conv2B but starting from a shot closer to the ground and pitching upward. The effect is that HMA_2 is seen powerfully from below as HMA_1 approaches.

**Conv3D**  A moving establishing shot that acts conversely to Conv3C; this one follows HMA_1’s walk until HMA_2 appears in the frame.
<table>
<thead>
<tr>
<th>Conv4</th>
<th>A blocking that leads into Conv2 with HMA_1 approaching HMA_2 from behind.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv4A</td>
<td>Same as Conv2F but for Conv4 blocking.</td>
</tr>
<tr>
<td>Conv4B</td>
<td>A medium wide shot of both from the perpendicular.</td>
</tr>
<tr>
<td>Conv4C</td>
<td>A medium close up shot of both from the perpendicular.</td>
</tr>
<tr>
<td></td>
<td>Conv4D</td>
</tr>
<tr>
<td>---</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>A very high crane shot from above.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Solo1A</th>
<th>Solo1B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A very long shot of the HMA from the place at which he is facing.</td>
<td>A medium shot of the HMA from the place at which he is facing.</td>
</tr>
<tr>
<td>Solo1C</td>
<td>A close-up the HMA from the place at which he is facing.</td>
<td></td>
</tr>
<tr>
<td>Solo2</td>
<td>A blocking with a human male adult (HMA_2) standing alone. Designed to be followed by Conv3 blocking, in which HMA_1 approaches HMA_2.</td>
<td></td>
</tr>
<tr>
<td>Solo2A</td>
<td>An extremely long shot of HMA_2 by himself.</td>
<td></td>
</tr>
<tr>
<td>Solo2B</td>
<td>A medium shot of HMA_2 by himself.</td>
<td></td>
</tr>
</tbody>
</table>
6.2 Character animations

The following are the gestures that are available for the stock avatars provided.

<table>
<thead>
<tr>
<th>Sarah (and other avatars built on PawnArtGirl)</th>
<th>Jeeves, Max (and other avatars built on PawnITGuy)</th>
<th>Chuck (and other avatars built on Chuck)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• HeadTurnL</td>
<td>• Idle_Rest</td>
<td>• gesture_beckon</td>
</tr>
<tr>
<td>• HeadTurnR</td>
<td>• IdleNeck</td>
<td>• gesture_cheer</td>
</tr>
<tr>
<td>• IdleEarring</td>
<td>• IdleNuckles</td>
<td>• gesture_halt</td>
</tr>
<tr>
<td>• IdleHip</td>
<td>• IdleGloves</td>
<td>• gesture_point</td>
</tr>
<tr>
<td>• Idle_Rifle</td>
<td>• Idle_Rifle</td>
<td>• Gesture_Taunt01</td>
</tr>
<tr>
<td>• Pickup</td>
<td>• Pickup</td>
<td></td>
</tr>
<tr>
<td>• Push</td>
<td>• Push</td>
<td></td>
</tr>
<tr>
<td>• Shove</td>
<td>• Shove</td>
<td></td>
</tr>
<tr>
<td>• Snowball</td>
<td>• Snowball</td>
<td></td>
</tr>
<tr>
<td>• TracySitTypeRight01</td>
<td>• IdleStretch</td>
<td></td>
</tr>
<tr>
<td>• TracySitTypeLeft01</td>
<td>• Throw</td>
<td></td>
</tr>
<tr>
<td>• TracySmoke01</td>
<td>• gesture_beckon</td>
<td></td>
</tr>
<tr>
<td>• TracyFlirtReact01</td>
<td>• gesture_cheer</td>
<td></td>
</tr>
<tr>
<td>• TracyGossip01</td>
<td>• gesture_halt</td>
<td></td>
</tr>
<tr>
<td>• TracyReactDismissive01</td>
<td>• gesture_point</td>
<td></td>
</tr>
<tr>
<td>• TracyReactCoy</td>
<td>• Gesture_Taunt01</td>
<td></td>
</tr>
<tr>
<td>• TracyReactGoAway</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyReactGasp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyReactDisappointed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyReactDismissive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyReactAgree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyReactShrug</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyIdleLookatFloor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyIdleReactInsult</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• TracyPhone</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. Extending the UT 2004 Virtual Backlot

Extending the virtual backlot is easy – you only need to use the map editing software that comes free with Unreal Tournament 2004 (c:\UT2004\System\UnrealEd.exe). You can add new places to the map and you can add new characters to be used in your movies.

7.1 Making New Places

To make new places, edit c:\UT2004\ALERT\maps\ALERTbacklot.ut2. Use the standard map editing techniques to carve out new geography and/or import static meshes. How to do this is beyond the scope of this document.

Note that Cambot and MST assume that all areas in which scenes can be shot are at the same z-value. The ground must also be perfectly flat. The ground must be flat so that a character’s head is always in frame. In the example map provided, there are two places: a city street and a portion of Georgia Tech campus. We have placed an invisible plane in the map on which characters walk (characters float a bit above the ground). When you carve out new places, make sure that the z-value of the ground is the same as that of the other places.

Once you have new places in the map, you need to let Cambot/MST know about them. Edit or create a new world XML file. Boxes must represent areas that are open and do not have objects that can occlude a camera shot or avatar movement. There can be any number of boxes, which can be non-contiguous. However, if two boxes are meant to be contiguous, they must overlap slightly.

7.2 Making new Characters

The system comes with a small set of prescribed characters. However, you can easily add new characters. All avatars are pre-placed in the virtual backlot map in an area out of the way. There are two techniques for adding new characters: skinning and modeling. Avatars consist of bones, model, and skin (texture).

7.2.1 Adding new characters via skinning

Skinning is a technique whereby an existing avatar model is given a new appearance by changing the textures mapped to it. The following steps will create a version of ArtGirl that has been reskinned.

1. Open UnrealEd and open the virtual backlot map.
2. Open the actor classe browser window.
3. Use the open dialogue to load “office.u”. This is where Cambot-specific character pawns are kept.
5. In the map, right-click on the ground and select “Add PawnArtGirl” here. You should see the pawn appear with default ArtGirl settings.
6. Double-click on the new pawn to open up its properties.
7. Expand the Events settings and change the Tag field to the new character’s name to “Lisa”. Now when you use Cambot or MST, you can refer to the “Lisa” character.

8. Expand the Display settings. Use the Scale3D to resize the character as appropriate. (This will likely require some trial and error. We recommend using MST to create a scene with the “Lisa” standing alone with a close-up shot. The character should appear on the screen more or less like:

![Character Image]

9. Change the skins values. Open the textures browser window and use the open dialogue to load “MBAGirl_T”. Select one of the textures then go back to the pawn properties. Expand the skins field and press the “use” button. Do this for all three textures.

10. Save the map.

7.2.2 Adding new characters with new model
You can create new characters using 3D studio max or other 3D modeling software. If you use the bone set that comes with UT2004 or can be downloaded from the Unreal Developer Network (http://udn.epicgames.com/Two/UnrealDemoModels.html), then you can add new characters without coding and recompiling the ALERT mod. Animations are attached to bones.

1. Use your 3D modeling software to create a new model and “rig” it with the existing bone sets (or create new bones but name them the same as existing bone sets, or create new bones with new bone names).

2. Use ActorX to import everything into UT2004 formats.

3. Follow steps 1-8 in Section 7.2.1, except:
   a. Use PawnITGuy or PawnArtGirl if you built off of the UDN demo model bones.
   b. Use Chuck if you built off the UT2004 model bones.

4. Change the model values. Open the textures browser window and use the open dialogue to load “MBAGirl_K01”. Go back to the pawn properties. Click on the mesh field and press the “use” button. Do this for all three textures.

7.2.3 Adding new characters with new model and animations
If you want to use 3D modeling software to create a character with unique bones, this gives you the power to create new animations as well. However, you will also need to provide a new pawn class. First you will need to acquire the ALERT mod source code. We recommend using the WOTGreal Unrealscript editing tool.

1. Use your 3D modeling software to create a new model and “rig” it with new bone sets (or use existing bone sets but with new animations).
2. Use ActorX to import everything into UT2004 formats.

3. Make a new package: create a new directory tree:
c:\UT2004\MyNewPackage\Classes\.


5. Make the new class extend the LBase.LPawn class, or any of the pawn classes in Office.

6. Override the base class as appropriate.
   a. Override the default properties so that Mesh points to the new model.
   b. Override the default properties so that Skins points to the new textures.
   c. Override the default properties so that AnimFiles array lists the names of all the animations that the new model can support.

7. Compile your new package.

8. Follow steps 1-8 in Section 7.2.1, except use MyNewPawn class instead.

8. Extending Cambot
Cambot comes with a set of blockings and shots. The most likely extension you may want to do is to add new blockings and shots. First you will need to acquire the Cambot source code. Blockings and shots are compiled into the system as individual classes in cambot/src/org/cambot/instances/blockings and cambot/src/org/cambot/instances/shots, respectively.

New blockings must subclass from Blocking.

New shots must subclass from Shot or one of its subclasses. Note that there are several abstract classes that make it easier to create new shots. For example, shots involving two characters facing each other should subclass from Conv1. The class, Conv1 OTS CU, provides many defaults useful for over the shoulder close-up shots.

9. Extending MST
If you create new shots or blockings for Cambot and wish to use them in MST, you must modify MST code; blockings and shots are hard-coded.

10. Writing your own virtual studio backlot
While we provide an Unreal Tournament 2004 mod as a virtual backlot application, Cambot and MST are designed to work with any virtual world technology that implements the following communication protocol. Consequently, we focus on the protocol and leave implementation details up to the developer.

When a game begins, the virtual studio backlot application should open up port 15003 for listening. Cambot/MST connects to the port and sends it commands as text strings. Each
string is followed by a newline. Commands have the syntax (key=value )+; that is, arguments that are key/value pairs separated by spaces. No key or value may itself contain spaces.

Every command must contain values for the following keys (i.e., these arguments):

- **action** – The value for action is the name of the command among the nine listed below. For example: action=CameraAdjust.
- **duration** – Execute the command the number of seconds represented by the non-negative float set for the duration key. For example, a fade may be set to last for 1 second, an animation may continue for 2.3 seconds, or a pan may take 4 seconds to complete. (The ExecuteGroup command does not require this argument.)
- **ID** – The value for ID is an alphanumeric string of your choosing, used to identify the command and distinguish it from other commands. If you are sending one command at a time, there is no point in distinguishing them, as the namespace should clear and there cannot be any ambiguity. However, when actions are cached into groups (see below), there can only be one action with a specific ID inside any particular group. When the virtual backlot finishes executing a command, it should send the value for ID back over the socket connection to the driving application.

In addition, every command may set any or all of these optional arguments:

- **group** – When the virtual backlot receives a command, it should immediately execute the command, unless a value is set for group. Similar to ID, group can be set to any alphanumeric string. If group is set, the virtual backlot should not execute the command until the special command ExecuteGroup is called for the group, at which time all the actions in the group are to be executed simultaneously. The purpose of groups is to ensure synchronization among multiple commands. There is a non-trivial time in socket communication and parsing, so commands that are sent adjacently may not be parsed and executed simultaneously. Instead, multiple commands should be put into the same group. When ExecuteGroup is then called, all the actions are fired off at once.
- **delay** – If delay is set to a non-negative float (e.g., 0, 2 or 3.6), the virtual backlot should wait the corresponding number of seconds before executing the command. Time is counted from when the command is “fired.” For commands that are not placed in groups, this is as soon as the virtual backlot finishes parsing the command. For commands that are placed in group, this is when the corresponding ExecuteGroup command is called. Hence delay may be used to schedule an action relative to other actions in a group, and entire scenes may be “planned out” in advance. If delay is not set, it is assumed to be 0.

There are three classes of actions: *Group* actions, *Camera* actions and *Character* actions.

**Group actions:**

1. **ExecuteGroup** – As foreshadowed above, ExecuteGroup can be used to execute all the commands previously declared as belonging to a particular group. Unlike every other command, ExecuteGroup does not require the duration argument. Every command in the group is executed according to its specific duration. ExecuteGroup has one
additional required argument, groupID. This should be set to the exact alphanumeric name previously specified as the group argument for all the commands one wishes to execute. For example, if four of nine commands are sent with group=g1, then the following command will fire just those four off at the same time:

```
action=ExecuteGroup ID=executeGroupG1 groupID=g1
```

**Camera actions:** There are four actions related to the camera:

1. **Wait** – The Wait action causes nothing to happen. This is useful in certain situations where “padding” is needed between actions. (Note, though, that other actions can be executed simultaneously with Wait.) It requires no further arguments. For example:

   ```
   action=Wait ID=wait1 delay=0 duration=4
   ```

2. **CameraFade** – This action causes the screen to gradually fade to or from a black screen. It takes one additional argument, `fadeType=(FADEIN|FADEOUT)`. For example:

   ```
   action=CameraFade ID=fadel delay=2 duration=4 fadeType=FADEIN
   ```

3. **CameraPosition** – The CameraPosition action is (often) used to place the camera at a particular location, rotation or focal length. Every CameraPosition command can only adjust one degree of freedom. To place the camera at a specific position, rotation and zoom, send seven of these commands, one for each degree of freedom, and execute them as a group. CameraPosition requires the following additional arguments:

   a. **adjustType** – This argument sets the degree of freedom being set. It must be one of these: XPOS, YPOS, ZPOS, PITCH, YAW, ROLL, and ZOOM.

   b. **value** – This argument is the position, rotation or focal length to which the camera should be set. For XPOS, YPOS and ZPOS, it must be set to the exact address (float) on the map where the camera should be positioned. For PITCH, YAW, ROLL, it should be a degree measurement (also float). For ZOOM, it should be an integer between 1 and 100.

   This example causes the camera to take 5 seconds to truck, pan, tilt and zoom from wherever it is to a particular 7-tuple:

   ```
   action=CameraPosition group=TwoShot ID=x duration=5 type=XPOS value=-2624
   action=CameraPosition group=TwoShot ID=y duration=5 type=YPOS value=-5184
   action=CameraPosition group=TwoShot ID=z duration=5 type=ZPOS value=-4864.00
   action=CameraPosition group=TwoShot ID=pitch duration=5 type=PITCH value=10
   action=CameraPosition group=TwoShot ID=yaw duration=5 type=YAW value=0
   action=CameraPosition group=TwoShot ID=roll duration=5 type=ROLL value=0
   action=CameraPosition group=TwoShot ID=zoom duration=5 type=ZOOM value=60
   action=ExecuteGroup ID=transitionToTwoShot groupID=TwoShot
   ```
Camera moves should be implemented with “slerp,” that is, spherical interpolation rather than linear. This causes the camera to ease into its move and ease out, achieving its maximum rate of change at the midpoint of the move. The camera should never rotate more than 180 degrees along any axis. In other words, it will always take the shorter rotational direction to get to its new position.

4. **CameraAdjust** – The CameraAdjust command works exactly the same as CameraPosition, except that the values represent relative changes to the camera’s current position, rotation and focal length, rather than absolute addresses. For example, this command causes the camera to tilt 10 degrees up from wherever it is, over 2 seconds:

```
action=CameraAdjust ID=tilt duration=2 type=PITCH value=10
```

This causes the camera to “cut” to a new shot where the camera is pointing 90 degrees to the left of where it was before:

```
action=CameraAdjust ID=pan duration=0 type=PITCH value=-90
```

This causes the camera to zoom in slowly:

```
action=CameraAdjust ID=zoom duration=10 type=ZOOM value=-20
```

In the context of CameraAdjust, duration sets the amount of time the camera takes to transition to the specified location, rotation or focal length. Setting the duration to 0 (or anything below .5) results in a direct cut; greater durations cause camera trucks, pans, tilts and/or zooms.

5. **SetCaption** – The SetCaption action should write text on the bottom of the two black bars creating a “widescreen” effect on the HUD. This may be used to indicate lines of dialogue, or for any other purpose. This action takes two additional arguments:

   a. **caption** – This is the text of the caption. Because there are no spaces allowed in any keys or values, Cambot/MST will place underscores in the place of spaces. Thus the virtual backlot should parse underscores and replace them with spaces.

   b. **withdraw** – This argument must be set to TRUE or FALSE. If it’s set to TRUE, then the caption is removed after the duration of the action has transpired. If it’s set to FALSE, then the caption will remain on the screen until SetCaption is called again with a replacement, regardless of the value of duration. For example, this causes an expository caption of “Georgia Tech Campus” to appear for 5 seconds and then vanish:

```
action=SetCaption ID=caption caption=Georgia_Tech_Campus withdraw=true duration=5
```

**Character actions**: The final group of actions governs the avatars that should be set up to be the characters in the machinima world. Each of the character actions requires the
following additional argument: character. The character argument should be set to the alphanumeric name of the avatar. The actions are:

1. **CharacterWalk** – This action causes a pawn to walk from its current location to a new location. The avatar should stop once it reaches the location. No pathplanning is necessary: avatars are expected to walk directly to the given destination.

   CharacterWalk requires the following additional arguments:

   a. $x$ – The X coordinate of the location to which the pawn should walk.
   b. $y$ – The Y coordinate of the location to which the pawn should walk.
   c. $z$ – The Z coordinate of the location to which the pawn should walk.
   d. $pitch$ – The degrees of “pitch” to which the pawn should rotate once it reaches the destination. For humans walking around, this should be 0.
   e. $yaw$ – The degrees of “yaw” to which the pawn should rotate once it reaches the destination. This is the axis that represents the direction that a human character faces.
   f. $roll$ – The degrees of “roll” to which the pawn should rotate once it reaches its destination. For humans walking around, this should be 0.

   For example, this command tells the character Cho to walk to a particular location and face in the direction represented by 100 degrees of rotation once he arrives.

   action=CharacterWalk ID=walk duration=2 character=Cho x=100 y=200 z=0 pitch=0 yaw=0 roll=100

   In this context, the duration argument is a timeout. If the time specified in duration expires before the avatar reaches its destination, the avatar should stop where it is.

2. **CharacterTeleport** – CharacterTeleport works exactly the same as CharacterWalk, except that the character instantaneously appears at its destination. In this case, the duration argument is ignored (but it is still required for the parser in the current implementation). For example, this command places the character Cho at a particular location on the virtual set:

   action=CharacterTeleport ID=beam duration=0 character=Cho x=100 y=200 z=0 pitch=0 yaw=0 roll=100

3. **CharacterAnimate** – This action invokes an animation on a character.

   CharacterAnimate takes the following additional arguments:

   a. $file$ – The name of the animation you wish to invoke on the character avatar.
   b. $loop$ – Set to TRUE if you wish the animation to play repeatedly until duration has expired; set to FALSE if you would prefer the character to play the animation once and remain idle for the balance of the duration.
c. time – Set to TRUE if you wish to adjust the speed of the animation so that it plays exactly once during the entire duration; set to FALSE if the animation should play at the “normal” speed of the animation.

For example, this command invokes the “nod” animation on the character of Cho as many times as will fit within a 10 second window:

```
action=CharacterAnimate ID=nod duration=10 character=Cho loop=true time=false file=choNod1
```

This example tells Cho to nod once, very slowly:

```
action=CharacterAnimate ID=nod duration=10 character=Cho loop=false time=true file=choNod1
```

**Acknowledgements**

Cambot development:

- Qun Cao
- David Elson
- Mark Riedl

MST development:

- Bryan Berry
- David Elson
- Aziel Ferguson
- Sabrina Heath
- Brian O’Neill
- Mark Riedl
- Brent Rowswell

Unreal art assets:

- William Daly
- Georgia Tech ImagineLab

Manual

- Mark Riedl