RoboRace

Jacob Kuenzel, Adhitya Chittur, and Devon Penney

April 14, 2006

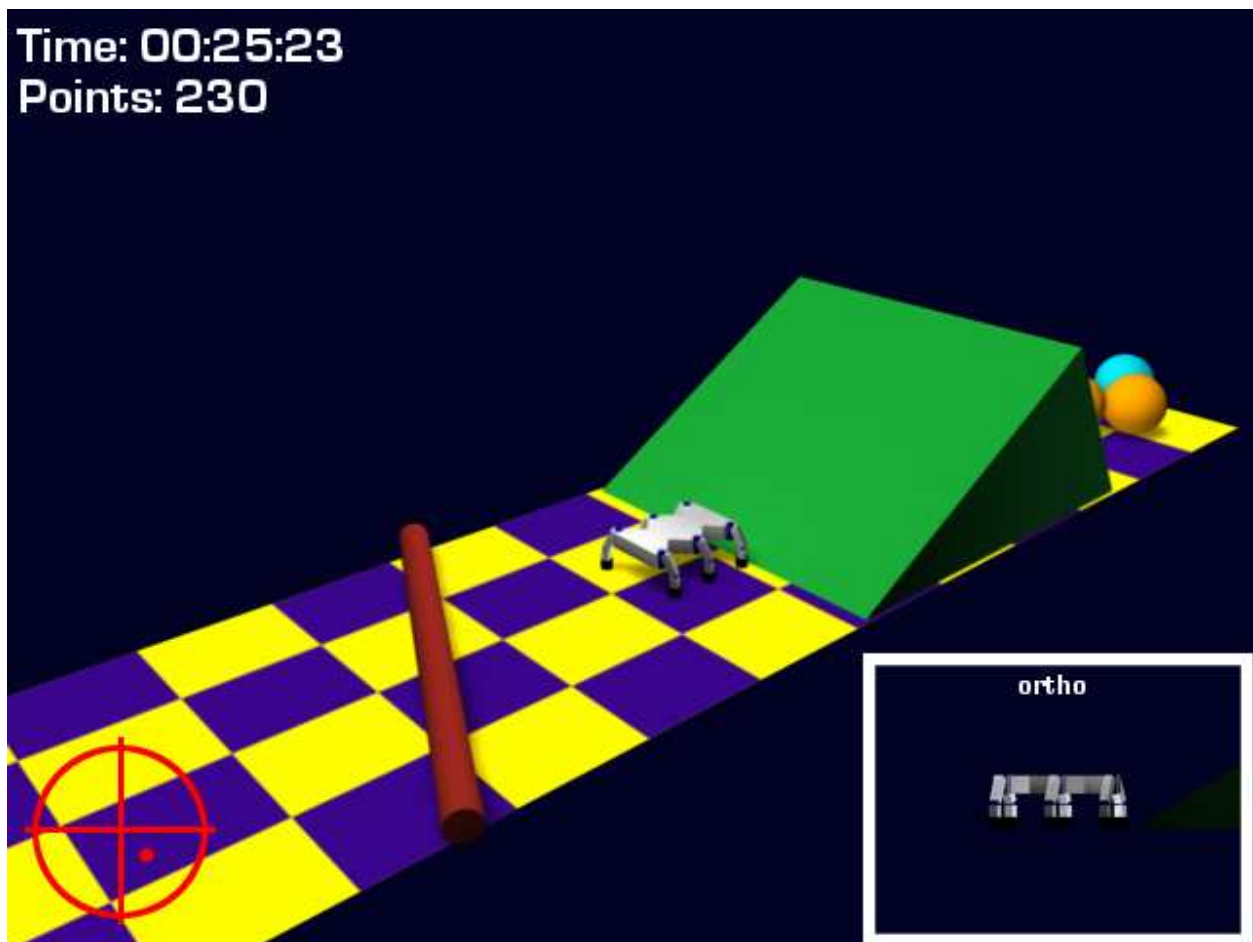## 1   Storyboard and Conceptual Images

Figure 1: A mockup of the in-game graphical interface.

Have you ever played the Ski Stunt Simulator? Probably not, but it is one of the most influential games from the last few years which has brought direct physical control to the world of gaming. The Ski Stunt Simulator (SSS) demonstrates the viability of direct two-dimensional control of a character in a two-dimensional world - our attempt is to bring this playability into a three-dimensional gaming environment. One big-name game that has incorporated similar direct physical control concepts is Fight Night, the boxing game for

consoles, in which the player directly controls the physical motion of the fists of their fighter. We believe that the playability of our game will be due to the emergent properties that arise from the mixture of physics simulation and novel user interaction. The style of this game raises questions: Can it be done in 3D? What games could this apply to?

We propose a physics-based racing game where users guide three-dimensional robots across a given terrain to the finish line. Users have the choice of several robots, each with unique physical characteristics. The robots may come in all shapes and sizes, and the user must choose based on both their own dexterity with the controller and the environment of the level. Users can also choose their control mechanism by assigning mappings between their input controllers and the actuators of the robot. Users will directly control the physics of the appendages of the robot, and the they are able to choose the scheme which best fits their style.

As in all racing games, the user will start with the robot at a fixed location, and must arrive at the goal location in the quickest time to win. We envision free-world levels - the maps will provide multiple paths, with different obstacles and challenges which test both the robot and player. Additionally, a point scheme for each race will be tallied, for additional penalties or bonuses - possibilities include damage to robot and stylish tricks. Finally, in a vs. computer scenario, both the player and the computer will have the ability to physically interfere with one another's play, providing an additional level of competitive play.

2   Previous Work

In [1], two interfaces for controlling characters in a 3D dynamics simulation are presented. The first uses a concept called "action palettes", in which the user is presented with a number of controls for possible actions (stand, run, jump). These actions are defined as either a set of target joint angles for PD controllers or as feedback-based balance controllers. Each action has associated with it 2 dimensions of control which the user may vary to achieve a desired result. While the interface in the paper is designed for non-realtime interaction, we feel that with some modifications and simplifications, it may be feasible in realtime.

The second interface presented in [1] is an attempt at controlling a 13-link rigid body humanoid character in a realtime snowboarding game with a typical game-pad. The authors note that "unlike traditional video game play, the stunts accessible through our interface need not be preconceived by the game author and can emerge as the product of the physics, the terrain, and the player skill." This freedom to invent novel moves for the character is one of the essential features of our game, and the results achieved by the authors while testing their interface are encouraging.

In [2], a number of interfaces are presented which are very specific to the character being controlled and the range of actions being performed. The authors point out the problem of controlling a character with a high number of degrees of freedom (DOFs) using an input device with a lower number of DOFs, and propose a number of solutions including the use of carefully designed one-to-many mappings between input and output DOFs, the use of keystrokes to complement and/or replace continuous control of some DOFs, and the use of state machines and other forms of automated control to assist the user. The authors also discuss the idea of checkpoints in time during the physical simulation which the user may return to if a mistake is made. Such a concept might be useful in our game if we find that it is too difficult to maneuver robots through the levels in a single attempt.

In both [3] and [4], motion capture data and optimization is used as a guide for creating new motions and poses of models. In the former, the authors make the observation that much of the difficulty in applying optimization to character poses is due to a problem in problem representation. Although a character may have many DOFs, useful and realistic motions involve a high degree of coordination, and can be well approximated using significantly fewer DOFs. The technique of Principle Component Analysis (PCA) is applied to motion captured data similar to the desired action to reduce the space in which optimization is performed. The user then specifies a set of constraints on the model through space and time, as well as a rough sketch of the motion. Optimization is then used to synthesized a realistic looking final result, though not in realtime.

In [4], a realtime optimization technique is presented. An objective function representing the desirability of a pose is derived from a set of training data. This function is then represented as a probability distribution function (PDF) of the set of all possible poses. In order to achieve optimization in realtime, the PDF is represented as a Scaled Gaussian Process Latent Variable Model (SGPLVM). New poses may then be created by constraining parts of the character and optimizing the objective function. One application the paper mentions for this technique is the ability to reconstruct realistic motion from incomplete motion capture data. It seems that by treating low DOF user input as incomplete data and applying this technique to it, it may be possible to achieve realistic and useful motions from our characters. If this technique were combined with the use of the keyboard to switch between various PDFs, it may be possible to create an intuitive an highly useful interface.

3   Project Software Engineering

3.1   Analysis Phase: Statement of Requirements

Our game will serve as a framework for experimenting with a number of different control methods. Therefore it is important that it is easy to add new control methods, and to switch between them in game. The game will be structured into number of modules: a physics simulation, an input and control module, a game logic module, and a user interface module. The following are our itemized requirements:

3.1.1   Physics

There are two main requirements to the physics engine necessary for the game: first, it must provide accurate simulation of articulated bodies, as this representation will be the basis of the player character. Additionally, the physical simulation must be able to run in real-time to provide a good user experience.
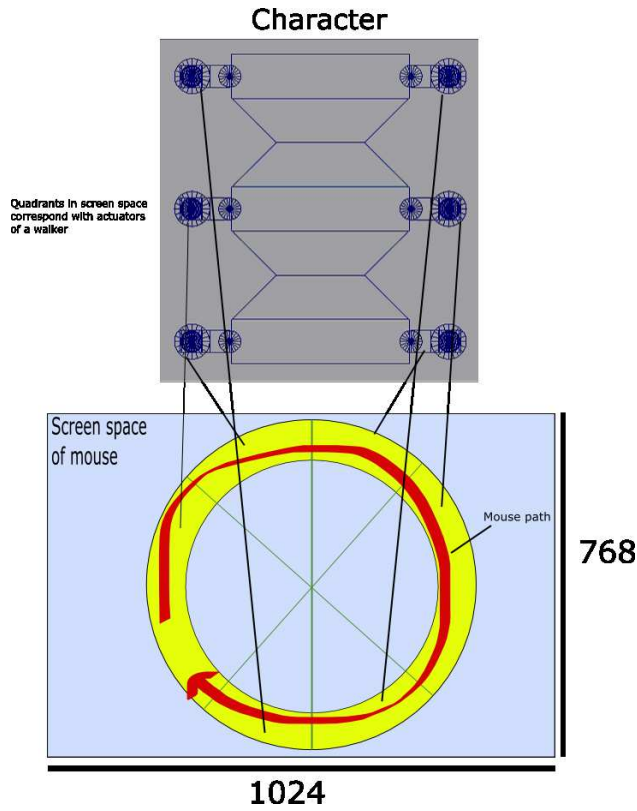
3.1.2   Input and Control

Figure 2: Example of the segmented input schem

The first requirement here is support for a number of different and novel control methods, including direct mapping, segmented mapping, and principle component analysis, as well as multiple input devices, including mouse, keyboard, and possibly gamepad. Each input device routed through a control method and then actuated on a specific robot comprises a *control module* - thus there may several control modules per robot. These input methods will provide users with several options on how to control their character, based on their style of play and effectiveness of the control module. The control module will also offer the user the ability to change various standard parameters of their input device, such as key bindings and mouse sensitivity. However, it is primarily the control methods we are concerned with.

The simplest method for controlling physics based characters is to map mouse motion directly to actuators. This is basically what is done with Ski Stunt Simulator. This works really well for the 2D case and very simple articulation models. However, it becomes significantly more difficult in 3D since models are more likely to have higher degrees of freedom. With two mice, it is possible to have four dimensional input, but there are few characters that could be adequately controlled by this. Hence, we may have to use methods such as action palettes or extensive automated control in these cases of controlling high DOF characters with direct input. It should be noted that direct mapping with little or no automated input could work well for a "pogo stick" style character which can lean and bounce for locomotion.

The basic idea in segmented input is to divide screen space so different areas correspond to activating different actuators. We have one possible example in the given drawing. Here, we are interacting with a multi legged walker. We segment screen space so the user makes circular motions, where arc regions of the circle correspond to activating actuators. There are possible extensions to this idea such as different radius circles mapping to higher forces (faster speeds). Also, one could make half circles or other gestures to indicate turning behavior. This segmentation method is most applicable to walkers with symmetric walkers

where walking behavior is achieved through sequential activation of locomotion actuators. Hence, the cyclic input we mention could provide an elegant solution for a user control method.

There are several ideas for PCA input, with some being much more complicated than others. The most important facet to this type of input is the availability of motion capture data for our purposes. It will be easy to come across humanoid data through databases. However, we will explore simulating PCA data through hand animation of robotic characters. An even more exotic idea would be to use genetic algorithms to learn walking behavior and then use that in PCA. . .not for this project.

Once we have motion capture data, the idea is to apply PCA to it. In the simple (potentially unintuitive) input method is to take the x and y mouse coordinates and use those as scalar coefficients to the first two basis vectors that describe the space of all motion capture data. By using a linear combination of these two basis vectors and coefficients, we can specify the desired pose. To some degree, this method of input will entail the user learning the appropriate gesture through the subspace which yields the appropriate motions for the character.

There is a much more interesting approach to controlling physics based characters with PCA. The idea is to use the method of Style IK to control the legs of a humanoid. Then, based on the new proposed motion of the legs, Style IK interprets the most appropriate pose of the character based on a probability distribution over all possible poses. Visit http://grail.cs.washington.edu/projects/styleik/ to view the videos.

We expect that some of the control methods we come up with will have fundamental limitations that make them difficult to use. To overcome these limitations, it must be possible to combine some automated control with the user's input. During our research we have encountered many papers which talk about the difficulty of manually balancing a humanoid character. By interpolating between user input controlling the legs and output from a module which calculates the closest balanced pose, it should be possible to avoid this difficulty.

As mentioned in the Previous Work section, high DOF characters often times perform motions in which many of the DOFs are highly coordinated. When a human reaches forward to grasp as object, for example, she will most likely lean her entire torso forward. By coming up with a list of common coordination schemes and allowing the user to select which scheme is being used during the game, it should be much easier to control a complicated character.

In an action palette scheme, it could become very tedious to be constantly switching which DOF is being controlled, particularly when performing repetitive actions. Walking, for example, could require the user to continuously switch which legs he is controlling. Detecting when a step is complete and automatically switching control to the legs on the other side of the robot could avoid this. More generally, an arbitrary action which has a well defined structure could be simplified by the use of a state machine. In a platform dive, for example, the following sequence of actions occur: stand, crouch, takeoff, aerial position, extension [1]. If the state machine were used to determine which DOFs of the character to manipulate, the user could simply tap one key to transition between states, greatly simplifying control.

3.1.3   Game Logic

The basic premise of the game is that it is a race, and thus the game logic is quite simple: it must only keep track of time, determine point deductions for the player, and determine the progress through the level.

3.1.4   User Interface

The pregame user interfaces will allow the following selections: (1) multiple robots with different locomotion schemes, (2) user changeable input modules, as specified in 2.1.2, and (3) choice of level.

The in-game user interface (Heads-Up Display) provides the user with information to realize the character's pose, position, and the controller's state. Additionally, pertinent game information such as time and points will be displayed.
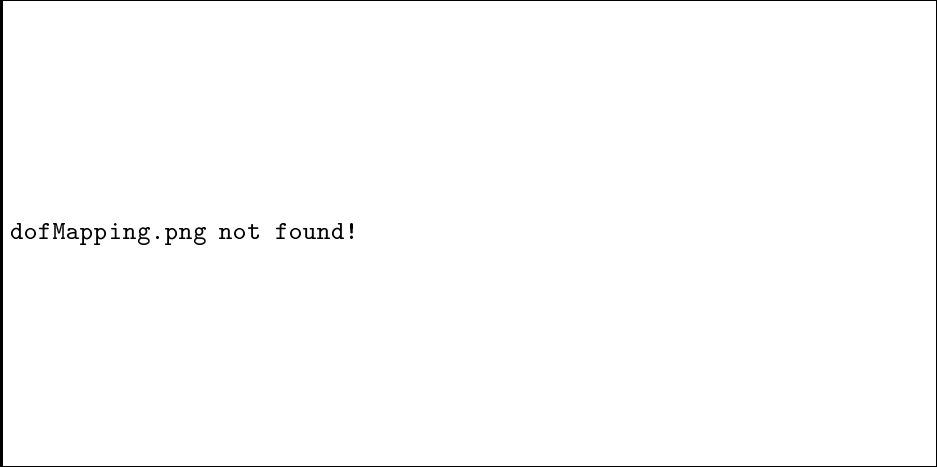
3.2 Mapping Input to Output

3.2.1 Definitions

Let $A = (a_1, a_2, ..., a_n)$ be the state of our input where a_i for $1 \leq i \leq n$ is the state of each degree of freedom. We will say $A \in X^n$ where X^n is a n dimensional space over the input. For example if we consider a mouse as our input, A is some (x, y) location of the mouse, and X^n is the screen space of all possible mouse locations, where $n = 2$. We define the output in a similar fashion. Let $B = (b_1, b_2, ..., b_m)$ be the state of our output where b_i for $1 \leq i \leq m$ is the state of each degree of freedom. We will say $B \in Y^m$ where Y^m is a m dimensional space over the input. For example, B is a current pose of a humanoid model where each b_i is the angle of an individual joint, and Y^m is the space of all possible poses.

3.2.2 Mapping

We want to map user input to character output. Formally, we define $f(A) = B$ where $f : X^n \rightarrow Y^m$. In this case, we have a direct correlation between the state of our input and output. For instance, our input is a mouse and we are mapping the mouse movement to the location of a cursor on the screen. We can also have the case where the state of the entire environment Z is also a factor for mapping input to output. Here we would have $g(A, Z) = B$ where $g : X^n \times Z \longrightarrow Y^m$. This, in effect makes context-dependent control for characters. An example is the use of automated control for characters. If we are trying to control a self-balancing robot, our input specifies a direction and speed to travel. However, we must alter the speed of the robot based on sensor input related to the orientation of the body.

dofMapping.png not found!

3.2.3 Dimensional Cases

Based on the number of degrees of freedom in our input and output, there are three cases to be considered for f from section three. First, consider the case where $m < n$ and we have a higher dimensional input than we do output. An example is using a mouse to scroll through a document. Scrolling is a one dimensional operation, which utilizes a single axis of the mouse movement. Next, we have $m = n$. The most simple example is the process of moving a mouse cursor on your desktop. Last, we have the most interesting example which is when $m > n$ and our output has a higher dimension than the input. Consider the problem of using a normal 2 dimensional mouse to control a humanoid character that has 60 degrees of freedom. Here there is no intuitive way to use our input scheme to control realistic humanoid movement. Also, in the case of controlling a humanoid form, we do not want to controll all 60 degrees of freedom independently. Rather, we would like to be able to have fine grain control the gross movement of the character in a simple and intuitive fashion.

3.2.4 Dimensional Reduction

The problem where $m > n$ as described above requires a way of modeling our output in a space with dimensionality equal to our input. Formally, we wish to find a space Y^n that approximates Y^m well. Also $\exists h : Y^n \longrightarrow W^m$ where $W^m \subseteq Y^m$. This effectively means we can reconstruct a subset of our ouput from our lower dimensional representation. Ideally, we would choose a lower dimensional representation that is intuitive for the output. For example, consider the problem of controlling a 60 DOF humanoid character for a walking behavior. We can choose Y^n be a space for controlling the legs of the character. Then, our function h would map the control in legs to a corresponding body position. In this case, our W^m is a space of all walking postures and Y^m is the space of all possible postures.

3.3 Design Phase

In light of the problems we have encountered with opal, we will be looking into moving to Dojo or possibly another platform. This design will need to be updated to account for this once we've made a decision.
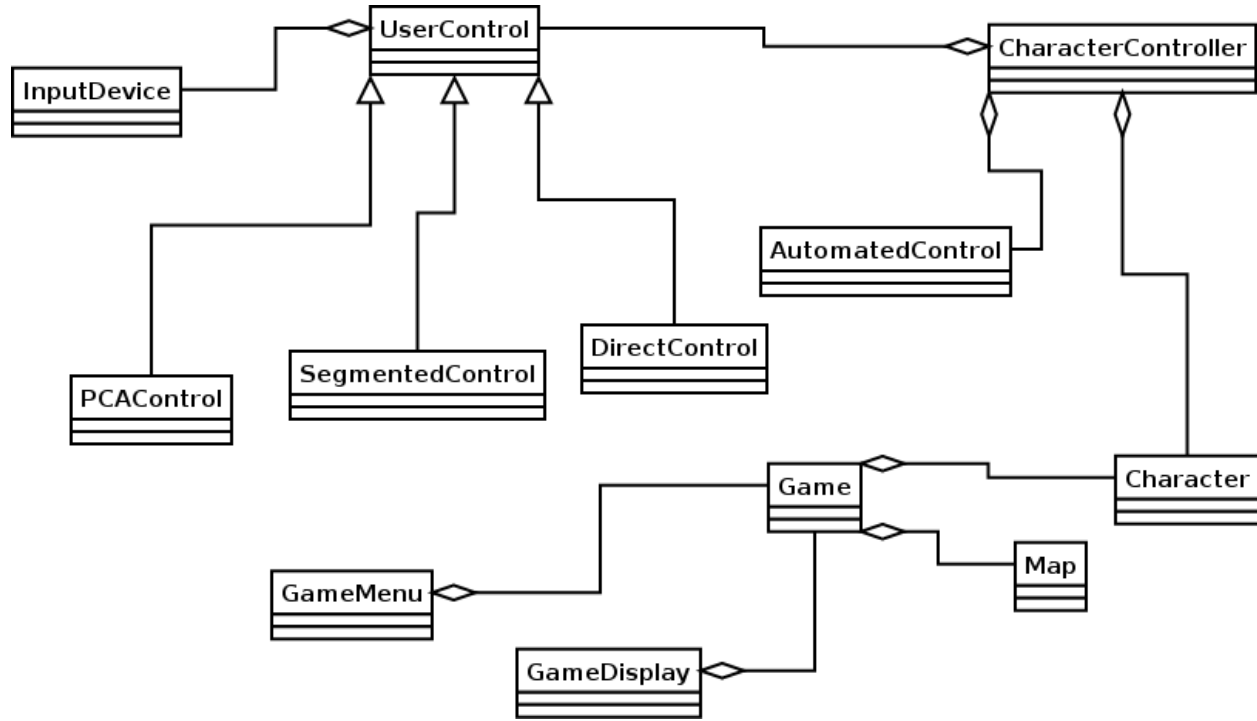

Figure 3: A class diagram showing containment and inheritance for framework classes.

3.3.1 Physics and Graphics

Class	Description
World	Contains the physics simulator, as well as a reference to all PhysicalEntities in the simulation. Has methods for loading a map, characters, etc.
PhysicalEntity	Represents an entity which exists in the World. Coordinates between ODE and Ogre.

For physics, we will use Open Dynamics Engine (ODE) due to its extensive use in the academic world for similar projects. It is a flexible system that has support for articulated bodies and servos. For graphics we will use the Ogre3D graphics engine. We will use an existing integration layer, OgreODE, to tie these systems

7

together. Additionally, to simulate articulated bodies, we will be using the Open Physics Abstraction Layer (OPAL), which is a high-level abstraction of physical components in ODE (see http://opal.sourceforge.net). OPAL contains abstractions such as Solids, Joints, Motors, and Sensors, which we believe will be sufficient support for simulating articulated bodies. Should we desire more abstractions than OPAL provides, we can add them to the OPAL framework ourselves.

3.3.2   Input and Control

The user interface must have support for multiple input devices (mouse and keyboard, and handheld controller if we can gain access to one) and multiple modes of input (e.g., direct mapping, segmented mapping, and principle component analysis). Each input module will be written specifically for the robot in use, the input device, and the mode of input. This will provide us with an extendable framework for experimenting with many different control interfaces.

Class	Description
InputDevice	Abstraction for mouse, keyboard, and possibly other devices.
AutomatedControl	Superclass for classes containing logic for automated control tasks (e.g., balance, coordinating movement of joints, choosing which appendage the user means to control).
UserControl	Superclass for classes implementing the various control methods we will be experimenting with (e.g., DirectControl, SegmentedControl, PCAControl). Constrains a reference to one or more InputDevice.
CharacterController	Contains a reference to a Character, UserControl, and possibly also an AutomatedControl object. Determines what the user is trying to do and actuates their character accordingly. This class is what we termed a control module previously, and each implementation of this class .

3.3.3   Game Logic

The game logic module will be fairly simple due to the nature of the game. It will primarily serve as a container for the game level, the player character, and the input control. It will also keep track of the race time and quantities such as damage points.

Class	Description
Game	Represents a game in progress. Has a timer and manages a trigger for detecting when the user has reached the end of the level. Contains references to Map and Character.
Map	Represents a single on-disk map file. Provides information such as spawn location and end location. Loads map at beginning of game.
Character	Represents a user controlled character. Keeps track of damage points, provides an interface for actuation.

Additionally, the game logic will handle the few event triggers which will be necessary for the game: The primary two triggers we require are a timer trigger, for starting the race, and a proximity trigger, to determine the finish of the race. If we have time, we may implement more triggers to make the levels more interesting, such as collision triggers and special-move triggers.

3.3.4   User Interface

The in-game user interface will, in addition to displaying quantities related to the game goal (e.g., time, points), will also provide state-space information to the user about their robot and control device. The heads-up displace will contain a planar representation of the character's current pose as well as a planar representation of the controller's current state-space and movement commands.

Class	Description
GameMenu	Responsible for the menus that the user is presented with when the game is started. Allows for selection of robot and map, as well as for configuration of controls. Instantiates a Game object to start a race.
GameDisplay	Responsible for the display in the game. Displays timer, damage points, helpful alternate views, and information specific to the current control method.

3.4   Implementation Phase

Implementation will progress according to the timeline in section 3. As stated in the design section, we will be using Ogre3D for graphics, ODE for dynamics, and OgreODE for integration between the two libraries. We plan on using the Dojo source code as a reference implementation for many of the features we would like our system to have, particularly the articulated model and motion capture code. Since Dojo is also based on ODE, it should be relatively straightforward to port parts of it to our system.

3.5   Testing Plan

Tests for this project can be split component-wise. The first basic test will be the realization of successful graphics and physics integration for display and actuation of the robot characters and their interaction with the environment. This will be accomplished both through unit testing and usage of the environment. The most important aspect of the graphics and physics is whether the movements pass the "look right" test.

The second test component will come from experimentation into the control module. This is partially a quantitative measure - control modules which allow the course to be completed in faster time will obviously have greater ranking. Damage accrued on the robot will also provide a second directly quantitative measure of the controller.

However, there is a very important qualitative aspect to the control module - the input must also be comfortable and addictively fun, which is highly dependent on the user. This is where the configurability of the controller will become an important factor, and it will simply be something that must be worked upon by inspection. We plan to quantize the qualitative aspects using a rubric akin to this for each control module, with each item rated on a 1-10 scale:

1. *Overall experience with control*

2. *Intuitiveness of control*: Did you find the controller difficult to use at first? Were you able to achieve proficiency in an acceptable timeframe?

3. *Ease of control*: Once you were familiarized with the controller, did you find it easy to control what you wanted to move and how it moved?

4. *Accuracy of control*: Once you were familiarized with the controller, did you find the controls to represent an accurate actuation of your desired input commands?

5. *Ability to move in a straight line*: Fast? Easy?

6. *Ability to negotiate turns*: Fast? Easy?

7. *Ability to negotiate obstacles*: Did this controller make obstacle negotiation intuitive? Fast? Easy?

In addition, we may employ external users for testing the controls and gaining more insight into what works and what does not.

4   Project Timeline/Milestones

Week	Accomplishments
1	OgreODE integrated and running with simple example, hopefully a humanoid. Get linux on Devon's laptop working with ethernet card for interactive demos in-class. *Devon now has Linux working on his laptop and we've decided to use OPAL instead of OgreODE. Humanoid demo using OPAL ragdoll blueprint was successful.*
2	Motion capture/physics working in OgreODE for humanoid. *Again, now using opal. Successfully parsing .asf and .amc files. Major problems with resulting skeleton's instability, however. Had a chat with Chad, will be considering a different platform and will also consult Pawel about his experiences with humanoid skeletons.*
3	Spring break.
4	Get up to speed on Dojo. Devon will get motion capture running and have playback in a physical simulation with a humanoid. Jacob will have a humanoid or hopper controllable via direct mapping (though the controls may not be optimal - this will require a lot of experimentation). Oddee will have a simple hopper implemented.
5	Devon will have preliminary dimension reduction techniques such as PCA working. Controls may be very unintuitive, but it will be possible to see that the humanoid is moving in response to input. Jacob will have direct mapping finished for humanoid and Oddee's hopper and will begin implementing a more complex robot such as a quadraped. Oddee will have initial segmented input working.
6	Devon will have more solid results involving PCA/StyleIK/etc. It should be clearer at this point whether the control scheme has potential. Jacob's more complex robot will be complete and controllable via direct mapping and segmented mapping. Oddee will complete segmented mapping and begin working on soccer game.
7	If dimension reduction is feasible, Devon will finish the interface and the humanoid will be controllable. Otherwise Devon will help Jacob iron any bugs out of control schemes and begin testing. Oddee will have soccer game completed and playable.
8	Finish testing and conduct an informal user study. Work out any last bugs and try to polish the demo.

5   Project Rubric: Expectations For Grading

Since we are experimenting with untried control methods, we have no way of knowing how effective or intuitive they will be. We believe that much of the work we will be doing will be implementing small variations on our ideas and/or tweaking constants to try to achieve reasonable usability. This type of work is inherently difficult to quantify—in order to demonstrate the effort we have put into the control methods, we plan on making all these variations accessible to the user. The more effort we put in, the more variations there will be. We will have all control techniques graded according to the testing plan, with external testers if possible.

A	At least one dimension reduction technique, segmented mapping, and direct mapping, all of which allowing for locomotion of a character. At least three variations on dimension reduction (PCA, Isomaps, PCA/Isomaps combined with direct control), at least two variations on segmented mapping (simple activation by segment, activation based on pattern of segment selection), and at least two variations on direct mapping (mapping to joint angles, mapping to joint positions and using IK). Demo involves a race and a soccer game. This soccer game will be rudimentary - the player will try to score goals against a multiple automated (motion capture driven, possibly

some rudimentary AI) opponents. Users can select which character to control in realtime during the soccer game.

B   Only one variation of dimension reduction and segmented mapping implemented. Two variations of direct mapping working (see above), Soccer game complete.

C   Only one variation of dimension reduction, segmented mapping, and direct mapping implemented. Soccer and race games are incomplete.

6   Selection and Approval of a Mentor

We've been running across a lot of papers citing Chad's work in our own research–will you be our mentor, Chad?

Response: Yes, I will be your mentor. I will ask Graham to back me up, just in case.

References

[1] Zhoa, P. and van de Panne, M. 2005. User Interfaces for Interactive Control of Physics-based 3D Characters. I3D: ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games. Washington, U.S.A., April 3-5 2005.

[2] Laszlo, J. F., van de Panne, M., and Fiume, E. Interactive Control for Physically-based Animation. Proceedings of ACM SIGGRAPH 2000, p.201-208.

[3] Safonova, A., Hodgins, J. K., and Pollard, N. S. 2004. Synthesizing Physically Realistic Human Motion in Low-Dimensional, Behavior-Specific Spaces. ACM Transactions on Graphics 23(3), SIGGRAPH 2004 Proceedings.

[4] Grochow, K., Martin, S. L., Hertzmann, A., and Popovic, Z. 2004. Style-based Inverse Kinematics. ACM Transactions on Graphics, SIGGRAPH 2004 Proceedings.