This documentation refers to the set of games developed for the robotic exoskeleton arm written in Microsoft Robotics Studio 2008 R2.

Code and document written by Sarah Richardson, sarahgrace89@gmail.com, Summer 2009.

1. Microsoft Robotics Studio

http://msdn.microsoft.com/en-us/robotics/default.aspx
(In order to run the games, you will need to download Microsoft Robotics Developer Studio 2008 R2 Express Edition and C# studio 2008 express edition)

MSRS has a large number of components related to robotics and simulation. The main one used by the virtual reality games is the simulation engine, which includes functionality to build an virtual reality world with physics. The robotic exoskeleton arm is represented in this world by a virtual arm which moves based on the data sent from the actual arm.

Unfortunately, neither the documentation nor the tutorials are very helpful in trying to understand MSRS. The beginner and advanced tutorials show you how to run previously created simulations that are encapsulated into manifests. For a basic tutorial of how to programmatically build a simulation environment and add basic entities to it, see the Tutorial project.

In general the physics engine will take care of any natural physics you want in a game, such as ball bouncing in a room (although bouncing objects with high enough restitution will actually bounce higher each time). However, there are often times when you want to manually move an item or set other physics properties related to it, which can be done using methods of the PhysicsEntity of the object. For example, you may want to create a ball that is moved based on the current position of the hand. Create the ball as a SingleShapeEntity with no mass and place it at a starting position:

```
SingleShapeEntity ball = new SingleShapeEntity(
                new SphereShape(
                    new SphereShapeProperties(
                        0,
                        new Pose(),
                        .05f)),
                    position);
ball.State.Name = "ball";
SimulationEngine.GlobalInsancePort.Insert(ball);
```

Every time you want to move the ball, call ball.PhysicsEntity.SetPose(Pose aPose) with the new position of the ball. The PhysicsEntity also includes a method SetLinearVelocity, while the ball.PhysicsEngine allows you to change the gravity of the item. (In this way you could give a ball mass but no gravity and then change the gravity to drop it).

When calling any method that changes the physics or the entity itself, it is best to use a Task. The simulation switches between running and updating, and if you try to update and entity while the simulation is running it can cause issues. Using a Task tells the simulation engine what needs to be updated so that it can be done at the appropriate time. For example, to change the position of the ball, you first need a method that does the move:

```
void MoveBall(Vector3 newPosition)
{
    ball.PhysicsEntity.SetPose(new Pose(newPosition));
}
```

Then, call the method using a Task:

```
Vector3 position = new Vector3(x,y,z);
Task<Vector3> move = new Task<Vector3>(position, MoveBall);
ball.DeferredTaskQueue.Post(move);
```

If you want to change other elements of the entity, such as the size or texture, the method SimulationEngine.GlobalInstancePort.Update(entity) must be used instead. Any method using Update should also be called through a Task.

2. Files:

All files for Microsoft Robotics studio and the projects I worked on are located in C:\Documents and Settings\Levi\Microsoft Robotics Dev Studio 2008 R2 Express\. The games I created are in the subfolder SRichardson. Below is an explanation of each folder and its contents.
SRichardson

- bkup - frequent backups of all projects
    - FormControlled - contains backups of the GUI controlled games, including the most recent versions
- Dev - contains the most current version of Games (all of the games in one application)
- Dev_separateGames - contains the last versions of the games as separate applications before they were pulled together
    - Working Demos - the old demos moved over from MSRS 1.5
- exo_models - mesh files created by Jay
- screen captures - images of the games
- videos - recorded videos of the games being played

Custom mesh files are in \store\media\custom and \store\media\man. The paths of these .obj files are referenced from the code using the path following \media\.

3. The Games project file:

Open Games.csproj to access the files and run the games.

- `Controller` starts the game and adds common entities to the environment.
- `Game` is an abstract class which each game object must implement. It contains some functionality and defines a handful of methods which either must or can be overridden.
- The Entities folder contains files for common entities such as the exoskeleton representation and the room.
- Each game has its own folder and namespace and consists of an object which extends `Game` and a `UserController` with game-specific controls
- `GameShell` can be used as a template to build other games. It is also run as a Game when no other game is selected (allows movement of the arm)

When the application is run, Controller is the first thing to be loaded. It creates the main GUI from MainForm and loads a UserController which allows the user to select a game to play. It also runs GameShell so that the visual arm can still be moved around in the virtual world using the exoskeleton arm. The property _game is the current game running on _gameThread; the Controller is only aware of the actual game when it is started. If the user clicks on one of the buttons to play a game, `GameShell` is aborted and `_game` is set to a new game based on the user's choice. At any point while playing the game, the user can click on "end game", which will alert the Controller. The controller will then call `Game.EndGame` (which removes all game-specific entities and aborts any threads) and begin to run `GameShell` again.

4. abstract class Game:

- MoveArm(float[] angles) - moves the visual exoskeleton based on an array of angles - should be used by all `Game`s
- AddToEnvironment and RemoveFromEnvironment should be used in all cases by the classes implementing `Game`, but SimulationEngine.GlobalInstancePort.Update may be used.
    - This allows all game-specific entities to be removed from the environment when the game is ended
- `EndGame` only needs to be overridden if the game starts its own threads which must be terminated before ending the game
- `PhysicsHandler` should be overriden if the game needs to be aware of physics collisions
- Refer to the comments for explanations of the remaining methods

5. Creating a new Game:

- Create a new folder (this will automatically put all new files into the Robotics.Games.<<*FolderName*>> namespace
- Add a .cs file called <<*game_name*>>Game.cs
    - Copy the text of GameShell.cs into the file
    - Change the namespace back to Robotics.Games.<<*FolderName*>>
- Add a new `UserControl` to the folder called <<*game_name*>>Control
    - Make the class private
    - Add a class property "<<*game_name*>> _game"
    - Pass in the <<*game_name*>> to the constructor and set the class property to it
- Change "`UserControl`" to your `UserControl` for the game class property and in the constructor (but not the get/set method)
- Add necessary environment objects
    - Add methods "`Add<<item>>()`" to create new objects
    - call these Add methods in the `PopulateWorld` method
- `Run` controls the position of the arm by calling `MoveArm`; add any other code here (in the `while(true)` statement) that needs to happen every time step
- `PhysicsHandler` is called by the controller when any physics collisions occur
    - Any entities for which you want contact notifcations:
        - first set `EnableContactNotifications = true` of the entity (e.g. `enitty.BoxShape.BoxState.EnableContactNotifications = true;`)

        - call `_controller.SubscribeForContacts(entity.PhysicsEntity)`
- Add methods to be called from GUI interactions as public methods in the game class
- in Controller
    - using Robotics.Games.<<*Namespace*>>
    - add an index to the public const int list
    - add case to `ChooseGame(int)`
    - add button to MainControl, when clicked call `ChooseGame`
- As a note, be aware that the man is facing in the negative x direction. This was something I did not really consider as an issue until it was too much trouble to turn him around. Therefore, to put something in front of the man the position must have a negative x value. To his left is the positive z direction, and to his right the negative z. (In several cases in the GUI the user can set positive x values for position and they are negated by the program).

6. Using the GUI:

All controls are loaded into the main Form window (called MainForm). For each game created, a UserControl needs to be created with the appropriate controls (as explained in section 4). Events in the GUI can call public methods in the Game class without issue. If you want to modify the GUI from the game (such as displaying current joint angles), you must do it in a thread-safe way. To modify a label in the GUI,

- In the UserControl class, create a delegate
    - `delegate void SetTextCallback(Label label, string text);`
- Create a private method for setting the text
    - ```
      public void SetText(Label label, string text)
      {
          label.Text = text;
      }
      ```
- Create a public method for resetting the text (where label is a Label in the UserControl)
    - ```
      public void SetLabelText(string text)
      {
          if (this.label.InvokeRequired)
          {
              SetTextCallback callback = new SetTextCallback(SetText);
              this.Invoke(callback, new object[] { label, text });
          }
              else
              {
                  this.label.Text = text;
              }
          }
      }
      ```

7. The Games:

- JointAngles
    - This is not so much a game as a tool. The user can move the arm around and the GUI will display all current joint angles.
- SingleJointMovement

    - Choose one of the seven degrees of freedom to move. All others will be visually locked and a semi-transparent plane will show the plane of movement for that joint. The GUI will display the maximum and minimum joint angles reached; these values will also be displayed visually by placing a red ball at the maximum and minimum points (the balls will move with the arm whenever the arm is exceding the previous maximum or minimum).
    - The plane is a VisualEntity with a .obj mesh file applied to it. This way there is no physics to interact with the movement of the hand.
- Pinball
    - Choose one of the seven joint movements to control a pinball game. As the user reaches a maximum and minimum joint angle, the pinball flippers will flip and return, respectively. This can be modified so that the left and right arm control the flippers separately.
    - The minimum and maximum joint angles can be modified based on the user's capabilities.
    - The original idea was to hit the ball with the virtual hand. However, it was very difficult to hit the ball with enough force to send it back up the table, even with a very small inclination. This method still tests the users joint movements and will be more playable for a handicapped patient.
    - There are several boxes on top of the table to keep the ball from bouncing out of the table. These boxes' meshes are set to a .obj file with nothing in it so that they are not visible. They work some of the time but occasionally the ball does not reflect and goes through the physical entity. However, when the ball is hit very had the it goes through even the "wood" edges of the table or over the top.
    - The flippers is a modified ExoEntity. The initial base joint and one ArmLinkEntity are used. In the Flipper constructor, the Swing1Mode is freed and the SwingDrive is set to Position so that the joint can be moved to the "hit" position and back to the "start" position. The damperCoefficient of SwingProperties affects how fast the flipper will move between the two positions (which affects with how much force the flipper hits ball upon contact).
    - Currently, the flippers both move when the user reaches a maximum angle and return to start when the user reaches the minimum position. This is a rather difficult way to play and does not really fit the rehabilitative purposes. There is also no way to score points, which would be a good addition.
- Reach
    - The user can reach out and touch a ball, making it fall to the ground.
    - Each ball is created with zero gravity. When PhysicsHandler is called (the balls are set up with physics contact notifications), the gravity of that specific ball that was contacted is reset to fall (using entity.PhysicsEngine.SetGravity).
- LineFollowing
    - The user must keep the ball on the line. If the ball touches the line it turns green and if it is removed it turns red.
    - There are two red balls, one at each end of the line. If the user touches a "goal" it turns green, but if he or she removes the ball from the line the goals turn red.
    - The line following can be done with one line, a set of lines in 2D or a set in 3D.
    - The single line works fine with the goals, but when there are multiple lines present there seems to be a delay in updating the colors of the various balls.

- Pong
    - The user can use one of eight control methods to move his racquet and keep the ball from hitting the front wall
        - The position of the hand relative to the middle of the table
        - The angle (with a known maximum and minimum angle) of one of the seven joints
    - The physics should allow the ball to hit the wall and bounce off without losing force by setting the restitution of both the ball and the wall to 1.0. However, the ball will simply hit the wall and roll along it. Instead, whenever one of the side walls is hit, the velocity is reset to the last velocity with the z value negated. The same happens when the ball hits either of the racquets with the x value negated.
    - When the ball hits either the front or back wall, the ball "disappears" using .PhysicsEntity.SetPose to a place outside the room. The appropriate player is given a point, which is displayed on the scoreboard.
    - When the ball is reset, it is placed right in front of the player's racquet and given a random velocity that will shoot it between the left middle and right middle of the table.
    - The scoreboard is a set of `SingleShapeEntity`s. The DefaultTexture of the blocks are changed based on a change in score or a message that needs to be displayed.
        - This is not a very aesthetic way to display the score, and perhaps not the most practical. There may be a better option.
- CircleMovement
    - Move the larger ball around the top of the cylindrical container to keep the bouncing ball from flying out.
    - The movement is based on the intersection of the line between the center of the cylinder and the current position of the hand and the circle of the cylinder (the center between the two walls). The ball is placed on the intersection of these two points. This actually allowed the user to place their hand at the center and move it in a very small circle, which was not the desired result. So the hand must be a certain distance from the center of the cylinder to interact with the ball.

8. Manifests and Deploying

The following explanations and instructions will allow you to create a portable .exe file to run the simulation on another computer. The computer must have Microsoft Robotics Studio 2008 R2 installed before the .exe file can be useful. Additionally, some games may run differently if the hardware configuration is different; the development machine had an Ageia PhysX card installed, which allowed certain things to run better than a computer using software physics. There are some other considerations that are noted below.

- Manifest -
    - This is an explanation of my understanding of manifests, but I do not fully understand how they work or what their purpose is.
    - All simulations are run based on manifests. Every time you compile and run your code the manifest is updated.
    - The location of the manifest file is specified in the project properties (Project -> Games Properties -> Debug). The file can be moved to a new location as long as this link is updated accordingly.
        - I moved the file to the SRichardson/Dev folder and changed the name to Games.user.manifest.xml

    - The contract link in the manifest xml file and the ControllerTypes.cs Contract class must be the same
- I changed the assembly name to "User.Games.Y2009.M08"
- To create a one-click shortcut to run the simulation (without compiling the code)
    - go to C:\Documents and Settings\Levi\Start Menu\Programs\Microsoft Robotics Developer Studio 2008 R2 Express\Visual Simulation Environment 2008 R2 Express
      (start -> MSRS -> Visual Simulation Environment)
    - Make a copy of one of the shortcuts
    - Open it and change the target link to "SRichardson\Dev\Games.user.manifest.xml" (the link to the manifest relative to the MSRS 2008 R2 folder)
    - this will create a shortcut to the manifest that can be loaded on the local machine with one click. This is not, however, portable to another computer.
- To deploy the simulation so that it can be run on another computer
    - One issue with making a portable .exe file is that it does not include .obj files used in the simulation. The meshes used in the project are in the MSRS 2008 R2/store/media/gamesMeshes folder. The links to .obj files are automatically relative to the /media/store folder, so it may not work to move the .obj files into the main Games folder. (Although this is not something I have looked into). The current solution is to copy the /gamesMeshes/ folder into the store/media/ folder of the other computer. There may be a better solution but I have no found it.
    - Run the DSS Command Prompt
    - dssdeploy /p /cv /m:"SRichardson\Dev\Games.user.manifest.xml" /s-Games.exe
        - go to http://msdn.microsoft.com/en-us/library/bb483014.aspx for documentation on this tool
        - /cv means it can only run in the current version (the same version the code was written in)
    - Games.exe can now be moved to any computer with MSRS 2008 R2 installed.
        - Copy the Games.exe file into the main MSRS 2008 R2 directory
        - Open it and click OK and also click OK to override various files in the directory (see warning)
        - WARNING: Be very careful running this on a computer with code you want to keep. Make a backup and double check the files it will be overriding. It is not recommended to run the .exe on your development computer
    - Follow the directions above for creating a one-click shortcut to run the manifest (which will now be in the same directory as it was on the original computer)

9. Suggestions for Future Work

- GUI changes
    - Make all number entry text boxes into MaskedTextBoxes that allow only numbers
        - MaskedTextBoxes may not be worth dealing with
        - Perhaps find another way to force the user to enter only numbers
    - Add images for selecting various games to the main window
    - Create a reusable UserControl for selecting a Vector3

    - Resize the window or somehow make the differences in space used by game controls more attractive
    - Display current joint angles when no game is running (or create a "game" that displays them and other values needed)
    - Add game descriptions/instructions
- Game changes
    - Pong: make the scoreboard more attractive. Use a mesh with number images. Images could look like those of a game scoreboard.
    - Pong: find experimental max and mins of comfortable movement of each joint and use these values for the default max/min values.
    - Pinball:
        - create a scoreboard for the simulation and add a point system so the user can score points
        - Thick (invisible) walls under the bottom wings of the table (to keep the ball from going "through" the wings)
    - Reset (pinball, pong, etc.) using a button on the device instead of the GUI
    - CircleControl: Instead of a position to drop the ball, choose the radius around the circle to drop it and calculate the position from that
    - JointMovement: adjust "planes of motion" to more accurately reflect possible minimum and maximum values.
    - Add functionality for the left arm (visual entity is in place but has no function)