**Hunt The Wumpus! 2007**

**wumpus@microsoft.com**

**Table of Contents**

[Introduction 4](#_Toc156132557)

[Distribution Lists 5](#_Toc156132558)

[School and Mentor Contacts 5](#_Toc156132559)

[Directions to Bellevue High School 8](#_Toc156132560)

[Directions to Interlake High School 10](#_Toc156132561)

[Directions to Inglemoor High School 12](#_Toc156132562)

[Directions to Newport High School 14](#_Toc156132563)

[Directions to Redmond High School 16](#_Toc156132564)

[Directions to Woodinville High School 18](#_Toc156132565)

[Leads/Teachers Pre-Project Meeting 20](#_Toc156132566)

[Course Syllabus 21](#_Toc156132567)

[Project Overview 22](#_Toc156132568)

[Student Application 23](#_Toc156132569)

[Assignments 24](#_Toc156132570)

[Assignment #1: Samples Modification and Project Creation 25](#_Toc156132571)

[Assignment #2: Understand Your Object 28](#_Toc156132572)

[Assignment #3: Data flow tables 30](#_Toc156132573)

[Assignment #4: Stubbed class 32](#_Toc156132574)

[Assignment #5:Start implementing functions in your stubbed class 33](#_Toc156132575)

[Assignment #6: Implement remainder of Object and Test Application 34](#_Toc156132576)

[Assignment #7: Testing and Content Expansion 35](#_Toc156132577)

[Assignment #8: Bug Bash and Creation of the PowerPoint Presentation 37](#_Toc156132578)

[Assignment #9: Practice Presentation 39](#_Toc156132579)

[Assignment #10: Final Presentation 40](#_Toc156132580)

[Grading Sheets 41](#_Toc156132581)

[Game Specification 44](#_Toc156132582)

[The Cave 44](#_Toc156132583)

[Hazards 45](#_Toc156132584)

[Wumpus 45](#_Toc156132585)

[The Player 45](#_Toc156132586)

[Warnings 46](#_Toc156132587)

[Fighting the Wumpus 46](#_Toc156132588)

[Money and Trivia 46](#_Toc156132589)

[Scoring 47](#_Toc156132590)

[User Interface 47](#_Toc156132591)

[Object Descriptions 48](#_Toc156132592)

[The Game Control Object 48](#_Toc156132593)

[The Graphical Interface Object 49](#_Toc156132594)

[The Map Object 49](#_Toc156132595)

[The Cave Object 50](#_Toc156132596)

[The Player Object 50](#_Toc156132597)

[The Trivia Management Object 50](#_Toc156132598)

[The High Score Management Object 51](#_Toc156132599)

[The Sound Object 52](#_Toc156132600)

[The Lazy Wumpus Object 52](#_Toc156132601)

[The Active Wumpus Object 52](#_Toc156132602)

[Interface Descriptions 53](#_Toc156132603)

[Notes on Extensibility 53](#_Toc156132604)

[The Game Control Object 53](#_Toc156132605)

[The Graphical Interface Object 55](#_Toc156132606)

[The Map Object 56](#_Toc156132607)

[The Cave Object 57](#_Toc156132608)

[The Player Object 57](#_Toc156132609)

[The Trivia Management Object 58](#_Toc156132610)

[The High Score Management Object 59](#_Toc156132611)

[The Sound Object 60](#_Toc156132612)

[The Lazy Wumpus Object 60](#_Toc156132613)

[The Active Wumpus Object 61](#_Toc156132614)

[Sample Code Overview 62](#_Toc156132615)

[Birthday 62](#_Toc156132616)

[Data Conversion Sample 63](#_Toc156132617)

[CD Library 64](#_Toc156132618)

[Whack-a-PM 65](#_Toc156132619)

[Test Application 67](#_Toc156132620)

[J# Crash Course 68](#_Toc156132621)

[Useful References 68](#_Toc156132622)

[Overview of Language Differences 68](#_Toc156132623)

[Lectures 70](#_Toc156132624)

[Wumpus Training 71](#_Toc156132625)

# Introduction

Wumpus Mentors:

In Spring 2004, a small group of Microsoft employees were welcomed into Newport High School in Bellevue with the intent of filling a niche that had been identified as a mutually interesting opportunity for the school and Microsoft to partner on. In specific, the generally excellent Newport High School AP Computer Science curriculum lacked an ability to challenge a select group of advanced students. For Microsoft, the opportunity to help in addressing this educational need was a timely one, presenting itself just as a group of employees involved as mentors in the high school intern program were looking for ways to extend the high school mentoring experience beyond simply the 10-week summer program.

Thus, after some initial brainstorming and planning, the “J# Development with Hunt The Wumpus” program was born – with an initial group of three Microsoft employees, six Newport High School students, and one very instrumental AP CS teacher.

This year, we welcome you aboard to share in the excitement of the substantially expanded program, which will reach dozens of students in Spring 2007!

We sincerely thank you for your interest and dedication to this program – and look forward to the challenges and opportunities that lie ahead.

*Evan Goldring*

*Louis Clausen*

*Harlan Husmann*

*Phillip Leslie*

*Jordan Naftolin*

Hunt The Wumpus Board of Directors

**Contacts**

## Distribution Lists

|  |  |  |
| --- | --- | --- |
| **DL Friendly Name** | **DL** | **Purpose** |
| Hunt the Wumpus @ Bellevue High School | htw-bell | Includes all leads+mentors for each particular school. |
| Hunt the Wumpus @ Inglemoor High School | htw-ingl |
| Hunt the Wumpus @ Interlake High School | htw-inte |
| Hunt the Wumpus @ Newport High School | htw-new |
| Hunt the Wumpus @ Redmond High School | htw-redm |
| Hunt the Wumpus Atlanta | htw-atl |
| HTW – Mentors | htw-ment | Includes all leads+mentors for the entire Hunt the Wumpus program. |
| Hunt The Wumpus | Wumpus | Wumpus Board of Directors |
| HTW – Leads | htw-lead | Includes all leads for the entire Hunt the Wumpus program. |
| Friends of Wumpus | Wumpfrnd | Includes various individuals that have asked to be included on major announcements (e.g. invitations to the field trip). |

## School and Mentor Contacts

### Bellevue High School

* Website: <http://www.bsd405.org/bhs>
* Address: 10416 SE Wolverine Way, Bellevue, WA 98004
* Leads:
  + Jordan Naftolin
  + Julian Selman
  + Jon LeVee
* Mentors:
  + Avram Wahba
  + Cody Ebberson
  + Jim Muliawan
  + Joe Chiu
  + Nate Muller
  + Reena Agrawal
  + Sacha Droz
* DL: htw-bell
* CS/Programming Teacher: <TBD>
* Principal: Mike Bacigalupi

### Interlake High School

* Website: <http://www.bsd405.org/ihs>
* Address: 16245 NE 24th, Bellevue, WA 98008
* Mentors
  + Bill Rainford (Lead)
  + Brent Bishop
  + Leslie Trowbridge
  + Brian Hudson
* CS/Programming Teacher: <TBD>
* Principal: Sharon Collins

### Inglemoor Senior High School

* Website: <http://schoolcenter.nsd.org/education/components/scrapbook/default.php?sectionid=19>
* Address: 15500 Simonds Rd NE, Kenmore, WA 98028
* Leads
  + Jeff Bogdan
  + Gildas Cheung
* Mentors:
  + Renato Martins
  + Jennifer Stepler
  + Tychaun Jones
  + Miguel Claudio
  + Matt Holle
* CS/Programming Teacher: <TBD>
* Principal: Vicki Sherwood

### Newport High School

* Website: <http://www.bsd405.org/nhs>
* Address: 4333 Factoria Blvd. SE, Bellevue, WA 98006
* Mentors
  + Louis Clausen (Lead)
  + Jiying Ren
  + Brian Railing
  + Dave Driver
* CS/Programming Teacher: Rod Thompson [thompsonr@bsd405.org]
* Principal: Patti Siegworth [siegwarthp@bsd504.org]

### Redmond High School

* Website: <http://schools.lwsd.org/rhs/>
* Address: 17272 NE 104th, Redmond, WA 98052
* Mentors
  + Molly Bostic (Lead)
  + Craig Jensen
  + Mike Scavezze
  + Lauren Lavoie
  + Kevin Weston
* CS/Programming Teacher: <TBD>
* Principal: Brian Hunter

### Woodinville High School

* Website: <http://whsweb.nsd.org/>
* Address: 19819 136th NE, Woodinville, WA 98072
* Mentors: None
* CS/Programming Teacher: <TBD>
* Principal: Vicki Puckett

## Directions to Bellevue High School

References for traveling to and from Bellevue High School:

|  |  |
| --- | --- |
| Driving Directions:  Microsoft to Bellevue  Estimated Travel Time 10 minutes | Miles |
| Depart 1 Microsoft Way, Redmond, WA 98052 on NE 39th St (North-East) | 0.1 |
| Turn LEFT (North) onto 159th Ave NE, then immediately turn LEFT (West) onto NE 40th St | 0.5 |
| Take Ramp (LEFT) onto SR-520 towards WA-520 | 3.2 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / Renton | 0.9 |
| At exit 13B, turn RIGHT onto Ramp towards N.E. 8th St. East/West | 0.2 |
| Keep RIGHT to stay on Ramp towards N.E. 8th St. West | 0.2 |
| Bear RIGHT (West) onto NE 8th St | 0.5 |
| Turn LEFT (South) onto Bellevue Way (NE) | 0.7 |
| Arrive 10416 SE Wolverine Way, Bellevue, WA 98004 |  |

|  |  |
| --- | --- |
| Driving Directions:  Bellevue to Microsoft  Estimated Travel Time 10 minutes | Miles |
| Depart 10416 SE Wolverine Way, Bellevue, WA 98004 on Bellevue Way (SE) (North) | 0.7 |
| Turn RIGHT (East) onto NE 8th St | 0.6 |
| Keep RIGHT onto Ramp towards I-405 | 0.3 |
| Take Ramp (LEFT) onto I-405 | 0.6 |
| Etc. |  |

|  |
| --- |
| **Detail Map of Bellevue Location** |
| **Overview Map of Bellevue Trip** |

## Directions to Interlake High School

References for traveling to and from Interlake High School:

|  |  |
| --- | --- |
| Driving Directions:  Microsoft to Interlake  Estimated Travel Time 5 minutes | Miles |
| Depart 1 Microsoft Way, Redmond, WA 98052 on 157th Ave NE [Microsoft Way] (South-West) | 0.5 |
| Bear RIGHT (West) onto NE 31st St, then immediately turn LEFT (South) onto 156th Ave NE | 0.5 |
| Turn LEFT (East) onto NE 24th St | 0.4 |
| Arrive 16245 NE 24th St, Bellevue, WA 98008 |  |

|  |  |
| --- | --- |
| Driving Directions:  Interlake to Microsoft  Estimated Travel Time 5 minutes | Miles |
| Depart 16245 NE 24th St, Bellevue, WA 98008 on NE 24th St (West) | 0.4 |
| Turn RIGHT (North) onto 156th Ave NE | 0.4 |
| Etc. |  |

|  |
| --- |
| **Detail Map of Interlake Location** |
| **Overview Map of Interlake Trip** |

## Directions to Inglemoor High School

References for traveling to and from Inglemoor High School:

|  |  |
| --- | --- |
| Driving Directions:  Microsoft to Inglemoor  Estimated Travel Time 20 minutes | Miles |
| Depart 1 Microsoft Way, Redmond, WA 98052 on NE 39th St (North-East) | 0.1 |
| Turn LEFT (North) onto 159th Ave NE, then immediately turn LEFT (West) onto NE 40th St | 0.5 |
| Take Ramp (LEFT) onto SR-520 towards WA-520 | 3.1 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / Everett | 5.1 |
| At exit 20B, turn RIGHT onto Ramp towards N.E. 124th St. | 0.2 |
| Keep LEFT to stay on Ramp towards N.E. 124th St. | 0.1 |
| Turn LEFT (West) onto NE 124th St | 1.3 |
| Turn RIGHT (North) onto 100th Ave NE | 1.3 |
| Turn LEFT (West) onto Simonds Rd NE | 1.1 |
| Arrive 15500 Simonds Rd NE, Kenmore, WA 98028 |  |

|  |  |
| --- | --- |
| Driving Directions:  Inglemoor to Microsoft  Estimated Travel Time 20 minutes | Miles |
| Depart 15500 Simonds Rd NE, Kenmore, WA 98028 on Simonds Rd NE (South) | 1.1 |
| Turn RIGHT (South) onto 100th Ave NE | 1.3 |
| Turn LEFT (East) onto NE 124th St | 1.0 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / Renton | 5.2 |
| Etc. |  |

|  |
| --- |
| **Detail Map of Inglemoor Location** |
| **Overview Map of Inglemoor Trip** |

## Directions to Newport High School

References for traveling to and from Newport High School:

|  |  |
| --- | --- |
| Driving Directions:  Microsoft to Newport  Estimated Travel Time 14 minutes | Miles |
| Depart 1 Microsoft Way, Redmond, WA 98052 on NE 39th St (North-East) | 0.1 |
| Turn LEFT (North) onto 159th Ave NE, then immediately turn LEFT (West) onto NE 40th St | 0.5 |
| Take Ramp (LEFT) onto SR-520 towards WA-520 | 3.2 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / Renton | 4.6 |
| At exit 10, turn RIGHT onto Ramp towards Coal Creek Pkwy. / Factoria | 0.3 |
| Bear LEFT (South) onto Coal Creek Pky SE | 0.5 |
| Turn LEFT (East) onto Factoria Blvd SE | 0.3 |
| Road name changes to 128th Ave SE [Factoria Blvd SE] | 0.1 |
| Arrive 4333 128th Ave SE, Bellevue, WA 98006 |  |

|  |  |
| --- | --- |
| Driving Directions:  Newport to Microsoft  Estimated Travel Time 14 minutes | Miles |
| Depart 4333 128th Ave SE, Bellevue, WA 98006 on 128th Ave SE [Factoria Blvd SE] (South) | 0.1 |
| Road name changes to Factoria Blvd SE | 0.3 |
| Turn RIGHT (North) onto Coal Creek Pky SE | 0.4 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / I-90 | 4.3 |
| Etc. |  |

|  |
| --- |
| **Detail Map of Newport Location** |
| **Overview Map of Newport Trip** |

## Directions to Redmond High School

References for traveling to and from Redmond High School:

|  |  |
| --- | --- |
| Driving Directions:  Microsoft to Redmond  Estimated Travel Time 14 minutes | Miles |
| Depart 1 Microsoft Way, Redmond, WA 98052 on NE 39th St (North-East) | 0.1 |
| Turn LEFT (North) onto 159th Ave NE, then immediately turn LEFT (West) onto NE 40th St | 0.4 |
| Turn RIGHT (North) onto Ramp towards WA-520 | 0.5 |
| Take Ramp (LEFT) onto SR-520 towards WA-520 / Redmond | 2.6 |
| Keep STRAIGHT onto Avondale Rd Ext | 0.3 |
| Road name changes to Avondale Rd NE | 1.2 |
| Turn LEFT (West) onto NE 104th St | 0.9 |
| Arrive 17272 NE 104th St, Redmond, WA 98052 |  |

|  |  |
| --- | --- |
| Driving Directions:  Redmond to Microsoft  Estimated Travel Time 14 minutes | Miles |
| Depart 17272 NE 104th St, Redmond, WA 98052 on NE 104th St (East) | 0.9 |
| Turn RIGHT (South) onto Avondale Rd NE | 1.3 |
| Keep LEFT onto Avondale Rd Ext | 0.2 |
| Keep STRAIGHT onto SR-520 | 2.1 |
| Etc. |  |

|  |
| --- |
| **Detail Map of Redmond Location** |
| **Overview Map of Redmond Trip** |

Directions to Woodinville High School

References for traveling to and from Woodinville High School:

|  |  |
| --- | --- |
| Driving Directions:  Microsoft to Woodinville  Estimated Travel Time 18 minutes | Miles |
| Depart 1 Microsoft Way, Redmond, WA 98052 on NE 39th St (North-East) | 0.1 |
| Turn LEFT (North) onto 159th Ave NE, then immediately turn LEFT (West) onto NE 40th St | 0.5 |
| Take Ramp (LEFT) onto SR-520 towards WA-520 | 3.1 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / Everett | 8.5 |
| At exit 23, turn RIGHT onto Ramp towards WA-522 / US-2 / Woodinville / Wenatchee | 0.2 |
| Take Ramp (LEFT) onto SR-522 towards WA-522 / US-2 / Woodinville / Wenatchee | 1.7 |
| Turn RIGHT onto Ramp towards N.E. 195th St. | 0.2 |
| Turn LEFT (West) onto NE 195th St | 0.1 |
| Turn RIGHT (North) onto 136th Ave NE | 0.2 |
| Arrive 19819 136th Ave NE, Woodinville, WA 98072 |  |

|  |  |
| --- | --- |
| Driving Directions:  Woodinville to Microsoft  Estimated Travel Time 18 minutes | Miles |
| Depart 19819 136th Ave NE, Woodinville, WA 98072 on 136th Ave NE (South) | 0.2 |
| Turn LEFT (East) onto NE 195th St, then immediately turn RIGHT (South) onto Ramp | 0.3 |
| Merge onto SR-522 | 1.1 |
| Take Ramp (RIGHT) onto I-405 towards I-405 / Bellevue / Renton |  |
| Etc. |  |

|  |
| --- |
| **Overview Map of Woodinville Trip** |
| **Detail Map of Woodinville Location** |

Leads/Teachers Pre-Project Meeting

After completing mentor training and receiving a school assignment, the following details are important ones to cover while meeting with the teacher for the first time:

* **Classroom Logistics:**
  + View classroom and working area
  + Find out if you will have access to: whiteboard/chalkboard, overhead projector, computerized projector, easil, isolated area of classroom.
  + Any Java books present in classroom?
  + Find out if teacher will be present during your lab sessions (encourage him/her to be - if at all possible)
  + Do students all have email access from the classroom?
* **Computer Setup:**
  + Confirm, or plan for, J#, PowerPoint, Word availability
  + Ensure - and verify - "debugging" permission is available (breakpoints needed)
* **Instruction Style Background** (context for deciding on your own instruction methods)**:**
  + Does teacher do lessons primarily verbally with just chalkboard/whiteboard? (most likely)
  + Does teacher ever lecture from PowerPoint?
  + Does teacher teach from book and have students follow along?
* **Scheduling Logistics:**
  + Find out how many days/week will students be able to work on this project.
  + 90%+ of each week is easiest to coordinate - but as a rule we always yield to AP coursework – whatever that means for us
  + Set up tentative schedule for classroom visit dates.
* **Student Pool and Background:**
  + Find out which students have Java experience.
  + Find out which students have PowerPoint experience.
  + Find out which of syllabus concepts are familiar to students.
  + Decide student/object pairings; give Game Control Object and GUI to relatively strong students.
* **Closing Messages:**
  + Reiterate that if students move to slow or too fast, we will adapt (we can always add more lectures/assignments)
  + Reiterate that if students need more lab-time help (e.g. J# help) we'll send extra folks to support in between main sessions
  + Give teacher your contact information and business card: we encourage you to include a mobile phone number in case of urgent matters

## Course Syllabus

**Unit 1 Concepts:**

* J# for Java Programmers
* Updating Sample Code

**Unit 2 Concepts:**

* Object-Oriented Design
* Working with Visual J#

**Unit 3 Concepts:**

* Requirements Gathering
* Scenario-Driven Design
* Data Flow Analysis

**Unit 4 Concepts:**

* Software Testing
* Object-oriented Design Process

**Unit 5 Concepts:**

* Software Testing
* Debugging

**Unit 6 Concepts:**

* Code Complete / Test Code Complete

**Unit 7 Concepts:**

* Code Reviewing
* Working on Enterprise Projects
* Multimedia Creation (Graphics and Audio)

**Unit 8 Concepts:**

* Demo and Presentation Skills

# Project Overview

**About the project**

In Hunt The Wumpus, you navigate a system of rooms within a cave in search of a monster named “The Wumpus”.  With bottomless pits, super-bats, and other hazards scattered throughout the cave, you may not survive the treacherous journey!  Succeeding in your quest requires a careful memory of the layout of the parts of the cave that you explore, and consideration of hints that you come across along your journey.

You may have heard of Hunt The Wumpus before. It is a classic turn-based computer game first developed in the early 1970s. We have recently revamped the original game to make it more exciting and add more complexity to the game. In developing this game, you will need to exercise the advanced Computer Science skills you have learned in Advanced Placement classes and elsewhere.

This game is designed to introduce you to a real life large-scale software engineering project where a team-based approach is necessary. You will use a modern object-oriented approach to implement the various pieces of the game, just as a commercial software company would. As a member of the team, you are fully responsible for your part of the project and you will work both collaboratively and independently until you and your teammates have completed all of the pieces. You will then work with your team to put the finishing touches on your game.

We encourage you to be creative while you exercise and sharpen your problem solving and computer science skills during the course of the 3 - 4 month long project. You will work hand-in-hand with a team of professional software developers from Microsoft. You will learn advanced concepts such as debugging, testing, bug fixing, User Interface programming, and multimedia programming as your game comes to life.

To implement the game, you will use the J# programming language which is very similar to the Java language used in many teaching environments. Although a sufficient background in programming is necessary to be a successful member of the team, specific knowledge of J# or Java is not required. Your C, C++, or C# skills easily translate into J#.

**About the program**

Microsoft started this program during the 2003-2004 school year as a way to reach the more advanced students in local area high schools who have a strong interest in computer programming. We continued the program during the 2004-2005 school year with continued success. This program fills a unique niche not currently found in any existing programs, including the AP Computer Science curriculum. Working on Hunt The Wumpus allows you to engineer on a real-world example by taking the computer science fundamentals you already know and apply them to solve a specific problem.

You will work in a group of approximately 5-6 people. We aim to provide at least a 3-to-1 student to mentor ratio so we can provide quality, timely, and hands-on help to you. Microsoft volunteers come to the classroom several times a month (typically weekly) for both teaching sessions and computer lab time where they assist the you with your programming assignments. The project is run as a series of one to two-week assignments that ultimately yield the finished product. Each assignment is graded using a consistent approach and scheme to what the school normally uses.

The program concludes with a day-long field trip to Microsoft where you will learn about various careers in the software industry, tour the inner workings of various Microsoft groups to see how games, Xboxes, portable media devices, cell phones, and other products are built. You will also be part of a presentation of your work. You and your teammates will have your worked judged and compete against the other schools participating in the program.

# Student Application

Name\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

E-mail address\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

High School\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Section 1 - Background**

List all math classes that you are taking or have completed at the High School or College level.

List all Science classes that you are taking or have completed at the High School or College level.

List all Computer programming or technology classes that you are taking or have completed at the High School or College level.

Tell us, in 250 words or less, why you are interested in this project.

**Section 2 – Sample of Work**

Along with this application, submit the source code for the longest and most complex original program you have written within the last year. This should be a program that you are particularly proud of. This can be a program you wrote for a class assignment, or one you wrote on your own. It is preferred that this program was written in C, C++, C#, or Java, but Visual Basic or scripting languages are acceptable as well.

**Section 3 – Programming Challenge**

Write a function that takes as input two numbers and prints out all of the prime numbers in between and inclusive of those two numbers. Write a program that uses this function to produce the list of prime numbers between: 0 and 1, 5 and 1930, and 7000 and 8000. Submit both the source code and the 3 lists of prime number within the ranges specified.

This program should be well commented and handle all error conditions appropriately. It is preferred that this program was written in C, C++, C#, or Java, but Visual Basic or scripting languages are acceptable as well.

# Assignments

The following assignments are designed to provide you with everything you need to be able to deliver a well-balance homework assignment. You *will* need to customize some of the homework assignments, with specific function names, etc.

## Assignment #1: Samples Modification and Project Creation

Due Date: \_\_\_\_\_\_\_

Point Value: 4

Background:

To help you learn to use J#, there are a number of samples that are provided in the Wumpus program. These samples are well-commented and cover a large number of different tasks that will be applicable in what you have to do.

For the first assignment, you will be familiarizing yourself with the samples by determining which methods in the samples do certain tasks. You will also familiarize yourself with creating projects in Visual J#.

### Assignment Task Part 1: Finding Reusable Code in the Samples

For each task in the list below, write in the sample, source file, and method name for where the code lives that would provide a good example to follow. To do this, you will need to try each sample to see what it does and then explore the structure of the samples.

Example: Loading High Score from a file

Sample: CD\_Library

Source: CDDBIO.jsl

Method: CD\_Collection::ImportCDCollection

Drawing a Wumpus to the screen

Sample: Source: Method:

Drawing the score on the screen

Sample: Source: Method:

Loading Cave information from a file

Sample: Source: Method:

Getting input from the user using a Text Box and Button

Sample: Source: Method:

(More on the next page!)

Getting input from the user with keyboard input

Sample: Source: Method:

Saving High Score information to a file

Sample: Source: Method:

Playing a sound

Sample: Source: Method:

Getting input from the user with the mouse

Sample: Source: Method:

Converting a String to an Integer

Sample: Source: Method:

### Assignment Task Part 2: Creating a Windows Project in J#

Hint: Part 2 will be easier if you've already completed Part 1.

1. Open Visual J#
2. Create a new project
   1. Select File 🡪 New 🡪 Project…
   2. Select “Windows Application”
   3. For the name of your project, use “WumpusTest”
3. You now see a Windows Form
4. Use the Toolbox to add a button, three text boxes, and two radio buttons
   1. The items will be used as follows:
      1. Two of the textboxes will accept numbers as input.
      2. The two radio buttons will be “+” and “-“.
      3. The button will perform the task selected by the radio buttons on the two input numbers.
      4. The third textbox will accept the output from the operation.
   2. After adding each item, selecting it will put the item’s properties into the Properties window.
   3. In the Properties window, in the “Design” section, change the name of each item to something descriptive
5. Double click on the button. This will take you to the J# function that will be run when the button is clicked.
6. Add code in this function to perform the proper operation.
   1. The code should first convert the text in the two input textboxes to integers.
   2. If the “+” radio button is selected, add the two numbers; if the “-“ radio button is selected, subtract the second number from the first.
   3. Convert the result to a string, and put this into the output textbox.

## Assignment #2: Understand Your Object

Due Date: \_\_\_\_\_\_\_

Point Value: 4

Background:

Hunt the Wumpus is a project that is broken up into a number of different objects that each can be worked on independently. Each of you will “own” one or more of these objects for Hunt the Wumpus. To be able to properly create your object, you will need to know how your object fits into the full game, including how it interacts with the other objects in the game and the tasks that it performs.

This assignment will give you the opportunity to understand both how the game works and how your object will do its part.

### Assignment Task:

1. Read the Hunt the Wumpus Specification.
2. Read the Object Description for your object.
3. For each of the scenarios in the next page, give up to three sections of the Wumpus Spec that control what your object does in that scenario.
   1. If you are unsure of what your object would do, put that in the list as, “I think my object will…” and make a best guess.
   2. If you don't think your object will do anything for the scenario, put in "Does not apply to my object" and don't put in any sections.
   3. Your lists do not need to be perfect: you will be graded on how much thought you’ve put into this, not on how correct your lists are.

A partial example for the Player Object:

|  |  |  |
| --- | --- | --- |
| Scenario | Spec Section | What My Object Does |
| User is in game, moves forward | Money and Trivia | Increases player's gold inventory by 1. |
| Scoring | Adds 1 to the number of turns user has taken. |
| User Interface | Gives the score, number of arrows, and gold count to be displayed. |

|  |  |  |
| --- | --- | --- |
| Scenario | Spec Section | What My Object Does |
| User wants to see High Score |  |  |
|  |  |
|  |  |
| User wants to start a New Game |  |  |
|  |  |
|  |  |
| User is in game, moves forward |  |  |
|  |  |
|  |  |
| User encounters Wumpus |  |  |
|  |  |
|  |  |
| User encounters bats |  |  |
|  |  |
|  |  |
| User falls into bottomless pit |  |  |
|  |  |
|  |  |
| User defeats Wumpus |  |  |
|  |  |
|  |  |

## Assignment #3: Data flow tables

Due Date: \_\_\_\_\_\_\_

Point Value: 4

### Background:

The purpose of this assignment is to better understand relations between the objects of the game. The data flow table and diagram should help you define interfaces between your object and other objects you depend on or/and that depend on you.

### Assignment Task:

Using the information from Assignment 1 and the exercise you went through in class, fill out the data flow tables that describe all data that your object uses.

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Called by or Calls This Object | Receives this Data | Returns this Data |
|  |  |  |  |

Draw data flow diagrams to illustrate the data flows and/or relations between the objects.

### Example Data Flow Table for Map Object:

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Called by or Calls This Object | Receives this Data | Returns this Data |
| Retrieve Hazard Warnings | Game Control | Current Room Number | String Array of Hazard Warnings |
| Move Player | Game Control | New Room Number (or direction to move) | True if move was valid  False if move was invalid |

Data flow tables for Object \_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Called by or Calls This Object | Receives this Data | Returns this Data |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Assignment #4: Stubbed class

Due Date: \_\_\_\_\_\_\_

Point Value: 4

### Background:

As a result of this assignment you will build a skeleton of your object that defines interface of your object exposed to the outside world. You will also create a test application that will help you build and test you object.

### Assignment Task Part 1: Stubbed class

Create a stubbed class (or classes) for your assigned object based on the data flow table for your object from Assignment #3. Requirements for the stubbed class (or classes):

* Is in correct J# syntax
* Contains complete member function signatures, including:
  + Function name
  + Argument name(s) and data type(s)
  + Return value data type
* Includes a dummy return line – for functions that have a return value
  + Example “return 0;” for a function that returns an integer
* Contains a comment inside the function body with:
  + The purpose of the function
  + The names of any other functions that it will call
* For all functions with return values, a comment indicating what the return value is for.

The stubbed class should also contain declarations for any member variables that are needed (that belong to the class – not the local member functions).

The class must build without errors (even though the function implementations don’t yet exist).

### Assignment Task Part 2: Test application

Add the class (or classes) to the application project (solution?) created in the Assignment #2. Add one button for each stubbed method to be able to invoke every one of the methods from the test application. Add text boxes for input parameters. Display output of methods where applicable in text boxes or message boxes.

Define formats of your data files (if you have any).

Bring your code to class and be ready to exchange with others. Create a copy of your project folder and name it <object>\_<date> for backup purposes and for future investigations.

## Assignment #5:Start implementing functions in your stubbed class

Due Date: \_\_\_\_\_\_\_

Point Value: 4

### Background:

The purpose of this assignment is to start implementing the actual functionality of your object.

### Assignment Task Part 1: Implement class functions (methods)

Implement two or more functions of your instructor’s choosing, and the constructor of your class. Your implemented functions must:

* Function independently of implementation of any other game objects.
  + If there are any other objects that you would need to call into to fully implement your functions, use stubs provided by the owners of the objects.
  + If for some reason you cannot use the stubbed class your implementation depends on, implement the missing/broken part of its functionality to yourself. Implement as much as is necessary to unblock your development.
* Correctly initialize all of the variable values
* Be clearly commented
* Be error-free and warning-free when compiled
* Handle error conditions that may arise

### Assignment Task Part 2: Update test application

The test application needs to be updated to invoke the two implemented methods if this hasn't already been completed.

Bring a list of issues you have with other student’s stubbed classes and your suggestions on how the issues should be resolved.

|  |  |  |
| --- | --- | --- |
| Object | Method | Description of the problem |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Assignment #6: Implement remainder of Object and Test Application

Due Date: \_\_\_\_\_

Point Value: 4

### Background:

The purpose of this assignment is to complete the rest of your object and prepare for its integration with the rest of the objects to create a playable game. You will need to update your test application along the way and you will need to test your object.

**Pre Assignment Task:**

Before starting this assignment, it is important to make a backup of your code from last week. This will help prevent any accidental data loss.

### Assignment Task Part 1: Implement remainder of Object

Building upon the function implementations that are complete for assignment #5, implement all remaining methods necessary for the class to be functional within the game.

Your completed methods must:

* Compile and be warning free
* Handle error conditions
* Initialize all variable values
* Be clearly commented

### Assignment Task Part 2: Implement remainder of Test Application

Building upon the test application from the previous assignment, expand the test application to include the ability to test all of the functions which are now exposed in part one of this assignment if this hasn't already been completed.

Each method should have:

* At least one button which invokes this method
* A way for the user to input an arbitrary value, such as an edit box
* A way to display the result of each method to the user
* Clear and descriptive comments
* No errors or warnings when compiled

### Preparation for Class:

Be prepared to share your code at the beginning of class next week and to bring a list of issues if your object has difficulty communicating with other objects

## Assignment #7: Testing and Content Expansion

Due Date: \_\_\_\_\_

Point Value: 4

### Background:

The purpose of this assignment is to verify the game is successfully integrated, the game is functional, and to add additional content. This does not mean the game is bug free but it should at least be playable.

**Pre Assignment Task:**

It is important to make sure you have a copy of the fully integrated game before leaving class.

### Assignment Task Part 1: Testing

With all of the game objects now fully integrated and complete, the game is now playable. Spend some time playing the game and then find at least five bugs in the game.

For all of the bugs you find:

* Describe the symptom of the bug
* Describe what you expect should happen

For two of these bugs also attempt to locate the source of the bug (for example: lines 20 to 25 of object.jsl)

### Assignment Task Part 2: Content Expansion

For this assignment, you will help create additional content. Communication with the object owners for Audio, Trivia, and Cave is critical to make sure you deliver content in the correct format.

**Audio Content (if Audio Object is implemented):**

Pick a theme and produce or create sound effects in the format specified by the Sound object owner for the following:

* Game Start
* High Score Dialog displayed
* High Score Dialog closed
* Super Bat encountered
* Bottomless Pit encountered
* Wumpus encountered
* Arrow hits the Wumpus
* Arrow missed the Wumpus
* Trivia answered correctly
* Trivia answered incorrectly

**Trivia Content:**

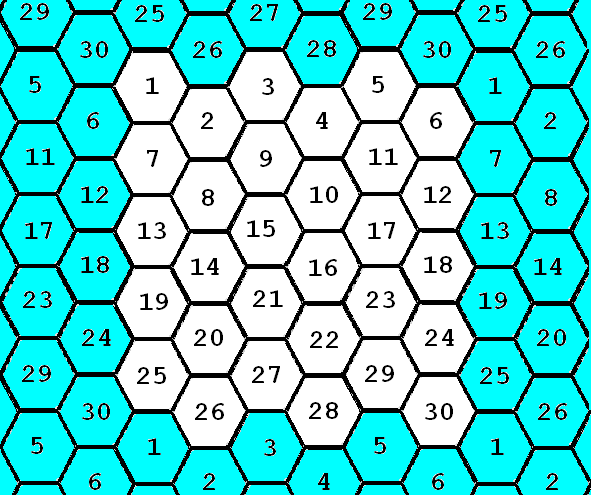
Create 15 trivia questions in the file format specified by the Trivia object owner.

**Cave Data:**

Create one cave map in the format given by the Cave object owner. Verify your cave file is correct by exchanging your cave with another student. This part of the assignment will not be accepted until another student has verified your cave data is correct.

Here’s a quick reference of the cave requirements:

* A cave consists of 30 rooms.
* Each room has up to three tunnels leading to other rooms.
* Each room must be able to reach any other room in the cave; no disjoint sets.
* The rooms on the edges may be connected to rooms on the opposite edge.



## Assignment #8: Bug Bash and Creation of the PowerPoint Presentation

Due Date: \_\_\_\_\_

Point Value: 4

### Background:

The first part of this assignment focuses on finding and fixing bugs in the game. Code testing, bug fixing, and ensuring the overall quality of the project are critical to the product life cycle.

Creating the PowerPoint presentation for the project allows the students and their mentors to provide information about themselves and the game and share that with the other teams.

### Assignment Task Part 1: Bug Bash

In this assignment, students must find bugs in their code and provide the necessary fixes before the project needs to be turned in. Students should also search for bugs in their teammates' codes.

Prizes and extra points will be awarded to students who find and fix the bug(s) that best fit(s) the following categories:

* Hardest bug to find
* Hardest bug to fix
* Most creative or inventive bug fix
* Most number of bugs found by a single student
* Most number of bugs fixed by a single student

### Assignment Task Part 2: Creating the PowerPoint Presentation

In this assignment, students are strongly encouraged to create their own PowerPoint slide with creative graphics and clip-art.

Each student on the team must create one PowerPoint slide containing the following information:

* Student's name and his/her background with computers
* Hobbies, job, interests, and activities
* Name of the object the student is responsible for and the purpose of that object
* Reason for wanting to work on this project
* Lessons learned while working on this project
* Most challenging part/aspect of the project
* Most enjoyable part/aspect of the project
* Least enjoyable part/aspect of the project
* Game elements you would add or modify if you had more time
* Bugs that you found or fixed that you are most proud of

Mentors are in charge of gathering the individual student slides and grouping these to form the team's project presentation. The team presentation must include slides for the following sections:

* [1 slide] A creative title slide (featuring the team's name)
* [1 slide per student] Student slides - introducing the students in the team
* [1 slide] Demo slide - time to show off your game!
* [1 slide] Q & A slide

## Assignment #9: Practice Presentation

Due Date: \_\_\_\_\_

Point Value: 4

### Background:

The purpose of this assignment is to allow students and their mentors to practice for their presentation, determine the order that each speaker will present, and to do a demo of their game.

This assignment provides the team an opportunity to work out any issues that may come up during the final/big presentation.

### Assignment Task:

In this assignment, students and their mentors should take their team PowerPoint presentation (from Assignment #8, Assignment Task Part 2) and go through the following tasks:

* Review each PowerPoint slide to make sure everyone on the team knows the order in which he/she will present and the order in which content will be presented.
* Once the order has been determined, each presenter should practice speaking in front of his/her team members using the slide that corresponds to that part of the overall presentation.
  + One of the mentors should introduce the team
  + Each student should present the information from his/her student slide
* The team should go through their game demo to catch and work out any issues that may come up during the final presentation.
* The team should make sure there is time for a Q & A at the end of the presentation.

Each team must practice their presentation at least three times.

## Assignment #10: Final Presentation

Due Date: \_\_\_\_\_

Point Value: 4

### Background:

This is it! Time to show off the result of all your hard work and share it with the other teams.

### Assignment Task:

In this assignment, students and their mentors will be doing a presentation about their team and their project in the Microsoft main campus.

Way to go, everyone!

# Grading Sheets

The following pages contain:

* A project-long grading template that should be duplicated and maintained for each student – by the mentors
* A single-assignment student-returnable grading sheet that should be duplicated and used to give each student their grade and feedback for each assignment

Your score values and weights may differ, based on discussion with the CS/programming teacher’s needs.

If, for example, additional assignments are added – the assignment point values need to be pro-rated such that the sum of the point values remains as 20 points.

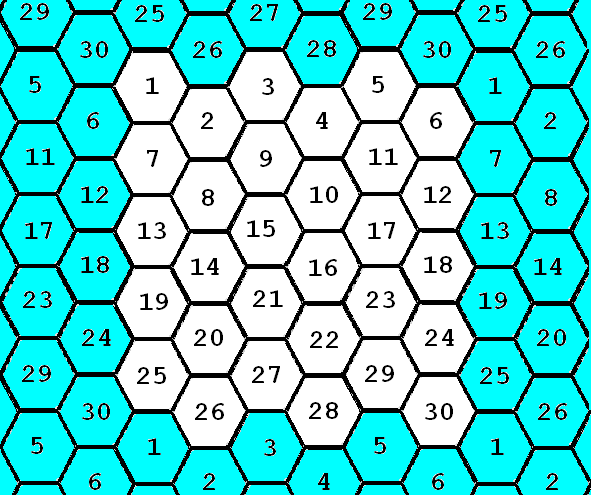
|  |  |
| --- | --- |
| **Grading Sheet for Student:** | |
| **Description** | **Point Value\*** |
| **Weekly Assignments** | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Assignment #** | **Earned** | **Maximum** | **Assignment #** | **Earned** | **Maximum** | | **1** |  | 4 | **6** |  | 4 | | **2** |  | 4 | **7** |  | 4 | | **3** |  | 4 | **8** |  | 4 | | **4** |  | 4 | **9** |  | 4 | | **5** |  | 4 | **10** |  | 4 | | **TOTAL** |  | | | | **40** |   (Notes: in general – the assignments should carry equal weight. If additional assignments are added, balance assignment weights as even as possible for a total of 20 points) |
| **Communication Skills**  *(including class participation and 1:1 discussions)* | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Session #** | **Earned** | **Maximum** | **Session #** | **Earned** | **Maximum** | | **1** |  | 1 | **6** |  | 1 | | **2** |  | 1 | **7** |  | 1 | | **3** |  | 1 | **8** |  | 1 | | **4** |  | 1 | **9** |  | 1 | | **5** |  | 1 | **10** |  | 1 | | **At-Large** |  | | | | 10 | | **TOTAL** |  | | | | **20** |   (Notes: adjust table based on number of classroom sessions; but do retain at least *some* at-large points – to be awarded based on how well students cooperate amongst themselves for the greater good of the project) |
| **Code Commenting / Readability and Documentation**  *(including variable naming, logical flow of code, etc.)* | |  |  |  | | --- | --- | --- | | **Session #** | **Earned** | **Maximum** | | **Initial Eval** |  | **10** | | **Final Eval** |  | **10** | | **TOTAL** |  | **20** |   (Notes: the initial evaluation should be after the students have completed the “two hardest functions” assignment; the final evaluation should be when the project is finalized at the end of the semester) |
| **Creativity / Code Sophistication**  *(e.g., an optimized algorithm yields high point value)* | |  |  |  | | --- | --- | --- | | **Session #** | **Earned** | **Maximum** | | **Initial Eval** |  | **4** | | **Final Eval** |  | **6** | | **TOTAL** |  | **10** |   (Notes: the initial evaluation should be after the students have completed the “two hardest functions” assignment; the final evaluation should be when the project is finalized at the end of the semester) |
| **End-Of-Project Presentation**  *(including both slides and presentation skills)* | |  |  |  | | --- | --- | --- | | **Session #** | **Earned** | **Maximum** | | **Content** |  | **8** | | **Presentation Skills** |  | **2** | | **TOTAL** |  | **10** | |
| TOTAL | 100 |

**Assignment #? Grading Sheet**

|  |
| --- |
| **Student:** ??? |
| **Object:** ??? |
| **Grade:** ???/ ??? (???%) |
| **Comments:**   * ??? * ??? |

# Game Specification

The Wumpus lives in a cave of 30 rooms. The rooms are hexagonal. Each room has up to 3 tunnels leading to other rooms. The rooms on the edges can be connected to the rooms on the opposite edge. The Diagram below demonstrates the cave layout and numbering (note that the player could travel from room 6 directly to room 30 or room 1, if room 6 connects to them).



## The Cave

You will need to create at least 5 different caves that the user can pick to play in. Each cave will have different sets of tunnels leading between the rooms such that in some caves you will be able to get between two adjacent rooms that you can’t move between in other caves. There should be no unreachable rooms in any caves.

## Hazards

Bottomless Pits - Two rooms have bottomless pits in them. If you go there, you fall into the pit. You can get out of the pit by getting at least 2 out of three trivia questions right. If you get out of the pit, you will be placed back where you started the game.

Super Bats - Two other rooms have super bats. If you go there, a bat grabs you and takes you to some other room at random. After the bat take you to a room, it will fly away to another random room.

No room will have more than one hazard. The Wumpus is not considered to be a hazard.

## Wumpus

The Wumpus is not bothered by the hazards (he has sucker feet and is too big for a bat to lift.) Usually he is asleep. Two things wake him up: your entering his room or your shooting an arrow.

If the Wumpus wakes, he sometimes runs to the next room at a random direction. If you happen to be in the same room with him, you have to fight the Wumpus.

## The Player

Each turn you may move, shoot an arrow, purchase more arrows, or purchase a secret.

*Move:* You can move one room at a time.

*Arrow:* You start with 3 arrows. If you ever run out of arrows without killing the Wumpus, you lose. Each arrow can be shot into an adjacent room. You aim by telling the computer which room you want to shoot into. If the arrow hits the Wumpus, you win.

*Purchase Arrows:* You can purchase 2 more arrows by getting at least 2 out of three trivia questions right.

*Purchase a secret:* You can purchase a secret by getting at least 2 out of three trivia questions right. The secrets range from not very useful to very useful. You can be told the room number where a bat lives, where a pit is, if the Wumpus is within 2 rooms of you, or the room number where the Wumpus is currently. However, you might also be told what room number you are currently in or the answer to a trivia question you have gotten wrong.

## Warnings

When you are one room away from Wumpus or hazard, the computer says:

Wumpus - I smell a Wumpus!

Bat - Bats Nearby

Pit - I feel a draft

## Fighting the Wumpus

If you wind up in the same room as the Wumpus you must get 3 out of 5 trivia questions correct. This only wounds the Wumpus, though. He doesn’t like to get beat in a fight and will run at least 2 rooms away, but can run up to 4 rooms away if he looses a fight. If he wins the fight, you loose the game.

## Money and Trivia

On every turn that you move through a tunnel, you will be given a gold coin and be told a piece of trivia. There are 100 gold coins to collect in total. Every time throughout the game that you have to answer a trivia question it will cost you a coin to attempt to answer the question. Whether you get it right or wrong, it costs you one coin and you will never be asked that question again before the end of the game. If you ever run out of coins, you loose.

## Scoring

The object is to kill the Wumpus in as few moves as possible. If you do, you will get 100 points! If you kill the Wumpus, your score will be computed with the following equation:

*100 points – N + G + (10\*A)*

N = number of turns

G = number of gold coins you have left

A = number of arrows you have left

The top 10 scores should be kept on file and include all of the information that made up the score (N, G, and A) as well as the player’s name and which cave was played.

## User Interface

At a minimum, the user interface should include a bird’s eye (looking from the ceiling of the room on down) view of the room you are in and an accurate depiction of the tunnels and which rooms they lead to. You will need to display at all times the number of coins the player has collected, the number of turns the player has taken, and the number of arrows the player has left.

You will also need to consider designing a user interface which will need to display secrets, hints, trivia information, trivia questions, number of correct trivia questions and number of questions needed for the current goal (at least when answering questions) an d allow for user input when necessary.

You will need to record the players name and let them choose which cave they want to play in before the game starts.

When not playing, the top 10 high scores should be displayed and pressing enter starts the game.

# Object Descriptions

Object purpose summaries:

* **High Score object:** Manages the high scores (including saving high scores and displaying a high score scoreboard).
* **Trivia object:** Manages the trivia questions for the game (including asking questions and loading questions from a file).
* **Map object:** Keeps track of the locations of the player, the Wumpus, and the hazards. Also keeps track of the cave and handles Wumpus movements.
* **Player object:** Keeps track of the player’s inventory and score. (This could be done by the same person doing the Map object).
* **Cave object:** Keeps track of which rooms in the cave are connected to which other rooms.
* **Graphics Engine object:** Displays the state of the current game (the current room, connected rooms, inventory, etc.).
* **Game Control object:** Handles user input (except for High Score and Trivia), coordinates all the other parts of the game.

Additional objects (these are only added when there are more than six students)

* **Sound object:** Plays sounds for the game, from a selection of different themes.
* **Lazy Wumpus object:** Controls the behavior of the Wumpus, providing it with more complicated behavior.
* **Active Wumpus object:** Controls the behavior of the Wumpus, but in a different manner than the Lazy Wumpus object.

## The Game Control Object

The Game Control Object coordinates all of the other pieces of the game:

* The Game Control Object keeps track of the game state (Currently playing a game, displaying the splash screen, displaying the high scores).
* The Game Control Object accepts and validates user input (that is, player commands and game control commands). The trivia and high score objects will manage their own user input.
* The Game Control Object interacts with the Map Object, the Player Object, the Graphical Interface Object, the Trivia Management System, and the High Score Management System.
* Some aspects of the Game Control Object should be exposed to the user through a “main menu” that allows the user to launch the game, display high scores, or exit.

## The Graphical Interface Object

The Graphical Interface Object is the object which does the actual drawing to the screen during the game. The tasks the Graphical Interface Object performs are as follows:

* Display on the game screen a representation of the room, including the hexagonal form of the room, with each edge illustrated as either a tunnel to an adjacent room or a wall blocking access to an adjacent room. Include an illustration of the player, any present hazards, the wumpus if present, and any additional graphics that add to the realism of the cave.
* Display on the game screen the player’s score.
* Display on the game screen the player’s inventory.
* Display on the game screen any hints based on the player’s room. (For example: I smell a wumpus).
* Display on the game screen all actions the player can take on the current turn.
  + Move
  + Shoot an arrow
  + Purchase arrows
  + Purchase a secret

## The Map Object

The Map Object tracks the locations of all of the objects in the current game. The tasks it performs are as follows:

* Store and interact with the cave used for this game
* Keep track of where the hazards are
* Keep track of where the Wumpus is. This includes controlling Wumpus behavior (that is, asleep, awake, moving).
* Keep track of where the player is
* Control arrow shooting.
* Give any necessary warnings
* Obtain secrets to help the player

## The Cave Object

The Cave Object keeps track of the cave, including data that describes the “connectivity information” for adjacent rooms. Adjacent rooms may either be connected by way of a tunnel, or not connected at all (separated by a wall). The tasks it performs are as follows:

* Read and parse map data from a file
* Stores connectivity information for each room in the map.
* Keeps an internal data representation of the map sufficient for obtaining and keeping track of all necessary information
* Exposes appropriate methods and/or attributes for other objects and the main program of the Hunt The Wumpus game.

## The Player Object

The Player Object keeps track of the player and all information associated with the player. The tasks it performs are as follows:

* Keep track of player inventory
  + Arrows
  + Gold coins
* Keep track of how many turns the player has taken
* Compute ending score of player

## The Trivia Management Object

This component of Hunt the Wumpus manages trivia questions that are used during gameplay, and secrets that help the user to progress through the game. The user will interact with it, when it is called by other game objects for the following:

* Seeking 3 trivia questions, of which 2 must be answered correctly, in an attempt to earn the right to purchase additional arrows
* Seeking 3 trivia questions, of which 2 must be answered correctly, in an attempt to earn the right to purchase a secret
* Seeking 3 trivia questions, of which 2 must be answered correctly, in an attempt to earn the right to be saved from a bottomless pit
* Seeking 5 trivia questions, of which 3 must be answered correctly, in an attempt to beat the Wumpus

Regardless of the number of questions that the *Trivia Management System* is asked to pose in a particular circumstance, and the number of answers that are required to be correct amongst the user’s responses in that circumstance, the *Trivia Management System*’s interface will be generic so that it can be called with a request to pose a specified number of questions and enforce the threshold of the number of questions that must be answered correctly. The caller of the *Trivia Management System* will be in charge of providing those numeric parameters.

The Trivia Management System will handle asking questions and getting the user response itself.

The Trivia Management System should be available throughout gameplay to be called as-needed.

## The High Score Management Object

This component of Hunt the Wumpus manages scores achieved by players that have won the game. The user will interact with this component in two ways:

* Storing a new high score
* Viewing existing high scores

The default high score data for the game should be the names of all of the participants in this project, with zero scores and arbitrary cave names.

The high score data will be exposed in a couple of ways: the user will be able to bring up the high score table through the game’s menu system, and the high score table will automatically be displayed after a player finishes a game.

At no time will there be more than 10 scores tracked by the *High Score Management System*. If a new score is submitted, that is good enough to be included in an already-full high score table, the new score will cause the lowest other score to be removed automatically.

## The Sound Object

The Sound object is responsible for playing sounds upon requests from the Game Control object. The tasks it performs are as follows:

* Reads list of sound files from its configuration file.
* Sounds files are grouped into sound themes (alternative sets of sounds).
* The Game Control object gets a list of available sound themes from the Sound object and selects (randomly or based on user’s input) an active scheme.
* The Game Control object calls the Sound object to play a particular game sound (player moves, player shoots an arrow, wumpus moves, triviea pops up, win, lose, etc.) from the selected theme.
* The Sound object opens and plays the sound file that contains the requested sound.
* There should be at least two sound themes.

## The Lazy Wumpus Object

The Lazy Wumpus object handles the movement of the Wumpus in the current game. The tasks it performs are as follows:

* Keep track of the current state the Wumpus is in (that is, asleep, awake, moving).
* If the player shoots an arrow and misses while the Wumpus is sleeping, the Wumpus wakes up and runs up to two rooms away from current position.
* If the Wumpus is defeated in trivia, it will run up to three rooms away.
* The Wumpus is slow and can only move one room per turn.
* If the Wumpus does not move for two turns, it falls asleep.

## The Active Wumpus Object

The Active Wumpus object handles a different set of rules of movement for the Wumpus in the current game. The tasks it performs are as follows:

* Keep track of the current state the Wumpus is in (\*that is, asleep and awake).
* Keep track of the number of turns.
* Every 5 to 10 turns the Wumpus will wake up and move 1 room per turn for up to three turns before going back to sleep.
* Every turn, there is a 5% chance the Wumpus will immediatelyteleport to a new, random location.
* If the Wumpus is defeated in trivia, it will run up to two rooms away per turn for up to three turns.

# Interface Descriptions

The following information is provided to help with the implementation of the objects.

## Notes on Extensibility

The Hunt The Wumpus program is designed to be very flexible, and can handle anywhere from 5 to 9 students. We recommend at most 7 to 8 students to take part in each project.

Here is a chart of what should happen with extra students (please note, map and player will be handled by the same person in each case):

|  |  |
| --- | --- |
| Students | Differences in object use |
| 5 | High score object is eliminated. |
| 6 | Default |
| 7 | Extra student implements sound object |
| 8 | * Extra student implements one of the Wumpus objects * Map student does not handle Wumpus behavior. Map student adds functionality: hazards up to two rooms away (that are interconnected) can be detected. (“You smell a Wumpus” plus “You faintly smell a Wumpus”). |
| 9 | Extra student implements the other Wumpus object |

## The Game Control Object

**Member Variables**

* A Map Object.
* A Graphical Interface Object.
* A Trivia Management Object.
* A High Score Management Object.
* A string which stores the name of the cave file that is used.
* An enumeration that indicates the current state of the object (Displaying Splash Screen, Displaying High Scores, or Playing a Game).

**Member Methods**

* A method which sets the name of the cave file to use. This method can use the OpenFileDialog .NET class.
* A method to start a new game. This method resets the Map object (using the cave file string), the Graphical Interface object, and the Trivia Management object. If no cave file has been specified, this method should call the method to get the cave file.
* A method which displays the High Scores.
* Various methods that accept user input. These should be implemented as events. With each user input, these methods should determine what to do with the objects. These methods should use the current state of the object (that is, the game should not accept a player movement if the splash screen is being shown).
* During a game, the following inputs should be accepted, through either mouse clicks or key presses:
  + Move player to room: This should call the Map’s method to change the player’s position. This method should then update the Graphical Interface object with all of the data and call its Render method. If there is a hazard or Wumpus in the new room, the game should wait an appropriate time and then take appropriate action.
    - Bottomless pit: Invoke a method of the *Trivia Management System* to give the user an opportunity to earn an exemption from the pit. If the player fails to get 2 out of 3 correct, end the game as a “loss”.
    - Super bats: Update the Graphical Interface object with the new position and available rooms, and call its Render method.
    - Wumpus: Start a Wumpus battle by calling the Trivia system to ask the player 5 questions. If the player gets 3 of those correct, the Game Control object will call the Map object to make the Wumpus run 2-4 rooms away (the game is not over until the player hits the Wumpus with an arrow) . If the player gets 2 or fewer correct, end the game as a “loss”.
  + Shoot arrow: Calls the Map object to shoot an arrow toward the selected room. If it hits the Wumpus, end the game as a “win”.
  + Purchase arrow: Call the trivia system to ask 3 questions. Update Graphical Interface.
  + Purchase a secret: Call the trivia system to purchase secret.
  + End the game.
  + Quit.
* A method that sends all relevant information to the Graphics object and has the Graphics object display it.
* A method that ends the game. If the game ended in a “win”, this should retrieve the current score of the game from the Player Object and call the High Score manager to potentially store the score, if it is within the top 10, and display the high scores.

## The Graphical Interface Object

**Member Variables**

* GDI+ Bitmap objects for any bitmaps that will be used to display information.
* A 6 element array which contains 3 room numbers with the position of each room number in the array corresponding to the direction of the room from the current room. The other 3 entries of the array will contain -1, to indicate that that direction is closed to the player. The indices of each array element imply the hexagon edge that the data element refers to; for example, the element at index zero might always apply to the left edge of the hexagon (from the birds-eye-view perspective).
* The player’s score.
* The number of arrows the player has.
* The number of gold coins the player has.
* A string that indicates which hazard is in the current room. If there are no hazards, the string should be null. This string is used when determining what hazard, if any, to draw when displaying the current room.
* An array of strings that contains the hints for the current room. If the same hint applies to a room for multiple reasons, for example if there are bottomless pits in two of the adjacent rooms, only one instance of the hint is stored (e.g., “I feel a draft”).

**Member Methods**

* An initialization method. The initialization method should load all needed bitmaps from files or resources.
* A method used to set the rooms. This method takes an array of room number as described above.
* A method used to set the player’s score.
* A method used to set the number of arrows the player has.
* A method used to set the number of gold coins the player has.
* A method that receives any hazards in the current room.
* A method that receives information about nearby hazards (for hazard warnings).
* A private method that takes a Graphics object and draws the map representation to the screen.

## The Map Object

**Member Variables**

* Room numbers to store the current location of:
  + The player
  + The Wumpus
  + The hazards
  + The starting point of the player
* The cave object

**Member Methods**

* A method which randomly picks the starting point of the player. This method could incorporate actually placing the player in the room.
* A method which randomly places all of the hazards in separate random rooms (no two hazards may be present in a single room)
* A method which randomly places the Wumpus in a room (initially placed in a room that has no hazard, but future moves of the Wumpus during the same game may cause the Wumpus to enter a room that contains a hazard).
* A method to kick off the process of reading in and storing a map. This method should take either the map number (1-5) or the filename corresponding to a map file. Its parameters should correspond to the behavior of the cave object.
* A method which, given a room number, returns a 6 element array holding the room numbers which can be reached via the tunnels in that room. Each element in the array corresponds to a fixed direction. If the room in a certain direction cannot be reached, the corresponding element in the array should be -1.
* A method to update each of the private attributes.
* A method or methods which return information indicating one or more of the warnings which need to be presented to the user, the hazard in the room with the user, or the fact that all is clear.
* A method to obtain the value of each of the private attributes
* A method which implements the action of the bats - moving the player to a random room (that has no hazards) and resetting that bat to wait in another randomly selected room that does not already have any other hazard present.
* A method which returns a randomly generated secret
  + Reveal a room number that contains a bat
  + Reveal a room number that contains a bottomless pit
  + Reveal a room number that contains the Wumpus
  + Reveal if the Wumpus is within 2 rooms of the player, as determined by walking distance (that is, by rooms that are connected).
  + Reveal the room number that the player is currently in

## The Cave Object

**Member Variables**

* A private set of arrays which store the room/tunnel data

**Member Methods**

* A method to read in and store a map. This method should take either the map number (1-5) or the filename corresponding to a map file.
* A method which, given a room number, returns a 3 element array holding the room numbers which can be reached via the tunnels in that room.

## The Player Object

**Member Variables**

* Number of arrows
* Number of gold coins
* Number of turns taken

**Member Methods**

* A method that removes an arrow from the player’s inventory.
* A method that removes any number of gold coins from the player’s inventory.
* A method that adds any number of arrows to the player’s inventory.
* A method that adds any number of gold coins to the player’s inventory.
* A method that indicates the player has taken a turn,
* A method that computes the player’s current score.

## The Trivia Management Object

**Initialization Activities**

This should occur when the player begins a new game, and will include the following:

* Read a file of trivia questions and answers, into an array in memory, from a file called TRIVIA.DB.
* Lines of the high score file are of the following format (Question; Answer):

“What city is the Space Needle in?”; “Seattle”

* If the file does not exist, the game should abort.
* The array into which the questions and answers are read must also contain a tracking mechanism for marking questions as “asked” to avoid repetition

**Trivia Q&A Responsibilities**

* When the *Trivia Management System* is called to pose a particularly number of questions and enforce a minimum number of correct answers, it should select each question randomly from the array of questions and answers.
* Before posing any questions, the constraints (maximum number of questions to be asked, number of questions must be answered correctly) that should be displayed for the user.
* After the question is selected, it should be displayed for the user and the user’s response should be received as input, and also marked as “asked” to indicate that it should not be selected again within the same game.
* The comparison between the user’s answer and the correct answer should be case-insensitive, and should confirm or deny the user’s answer, and indicate to the user whether the answer was correct or incorrect. The result of the comparison should also be stored in the array with the trivia questions and answers, so that the *Trivia Management System* can keep track of which questions were incorrectly answered, for which the answer may eventually be revealed to the user in the form of a secret.
* Once the user either satisfies the required number of correct answers, or exhausts the allowable number of questions (whichever comes first), the *Trivia Management System* should inform the user of whether or not they answered enough questions correctly. This result should also be indicated to the caller of the *Trivia Management System*.
* Regardless of the outcome of the questions and answers, since the game will charge the player one coin per trivia question, the *Trivia Management System* must indicate back to it’s caller the number of questions that were actually posed (the user can conserve coins if it answers enough questions correctly before reaching the question limit)

## The High Score Management Object

**Storing a New High Score**

When a player beats the Wumpus, the score will be computed by the *Player Object*, and that score and cave number will be passed to the *High Score Management System*, which will:

* Read a file of high scores, into an array in memory, from a file called HIGHSCORE.DB.
* Lines of the high score file are of the following format (Name, Score, Cave Number):

“Pablo Picasso”; 52; 3

* If the file does not exist, the system should assume that no high scores exist.
* Search through the entries and determine if the score that was passed in would be within the top ten.
* If the score would not be in the top ten, skip this step, otherwise:
  + Prompt the user for a “Player Name” for the high score.
  + Update the array in memory based on the new score, potentially causing the lowest score to be removed from the array completely.
  + Write out the new high score data to HIGHSCORE.DB.
* Display the high score table

**Viewing Existing High Scores**

* Read a file of high scores, into an array in memory, from a file called HIGHSCORE.DB.
* Sort the array of high scores in memory, from the highest score to the lowest score.
* Display the top ten scores (or less if the file does not contain ten), showing all three elements of the high score (Player Name, Score, Cave Number)

## The Sound Object

Member Variables

* Path to the object’s configuration file
* Available sound themes
* Paths to sound files for all sounds of all themes (loaded from the configuration file)
* Active theme

Member Methods

* A method which returns a list of available sound themes
* A method which sets an active theme
* A method which plays a requested sound

## The Lazy Wumpus Object

Member Variables

* The Wumpus state
* Number of rooms left to move

Member Methods

* A method which returns the number of rooms the Wumpus still has to move
* A method to set the number of rooms the Wumpus still has to move
* A method which activates the Trivia defeat behavior of the Wumpus, and returns the number of rooms the Wumpus will move
* A method which will set the Wumpus state
* A method which will return the Wumpus state

## The Active Wumpus Object

Member Variables

* The Wumpus state
* A turn counter
* Number of rooms left to move

Member Methods

* A method which returns the number of room the Wumpus still has to move
* A method to set the number of rooms the Wumpus still has to move
* A method which activates the Trivia defeat behavior of the Wumpus, and returns the number of rooms the Wumpus will move
* A method to check for random teleportation, which will return the new room location of the Wumpus
* A method which will return the Wumpus state
* A method to set the Wumpus state
* A method which resets the turn counter
* A method which increments the turn counter

A method which returns the turn counter

# Sample Code Overview

To help the students get ramped up with the different concepts they will need for the Wumpus project and to give us J# experience, we put together four different samples that each demonstrates a collection of important J# concepts.

## Birthday

Birthday uses a dialog box to prompt the user to enter their birthday. From this birthday, the sample calculates the user’s age. Perfect for use by law enforcement or liquor vendors.

### Core Concepts Demonstrated

This sample demonstrates the use of Forms in J#, including using text boxes for input and using static text boxes for output.

### Source File Enumeration

There is one main source file that can be edited for the Birthday sample, plus a few source files that are automatically generated by Visual J#.

* Form1.jsl: This is the file that implements the core functionality of the sample.
* Form1.resx: This file is automatically generated by VJ#, and contains information about the dialog box.
* AssemblyInfo.jsl: This file is automatically generated by VJ#, and contains information used by VJ# when building the binary file.

### Architecture Description

The Birthday sample contains one class: Form1. This class was autogenerated by VJ# to fully implement the sample dialog as created in the design view.

## Data Conversion Sample

The Data Conversion Sample demonstrates proper conversions between the various data types and the String data type.

### Core Concepts Demonstrated

This sample demonstrates the proper methods to convert data between the String data type and the various builtin datatypes.

### Source File Enumeration/Architecture Description

The Data Conversion sample contains one source file, Form1.jsl, which contains both the form layout and the code for data conversions.

## CD Library

The CD Library is a console-based application that allows the user to enter and save information about a CD collection. The information can then be loaded later and even modified.

### Core Concepts Demonstrated

This sample demonstrates the use of console input and output, which itself is not necessary for the Wumpus project but can be useful. More importantly, the CD Library demonstrates methods for saving information to a file and reading the information back in from the file, with techniques for parsing strings.

### Source File Enumeration

The CD Library sample follows the VJ# standard (also standard in Java) of using one source file for each class in the application. In this case, there are five different source files, along with a sample CD database file.

* CD.jsl: This file contains the implementation of the CD class.
* CD\_Collection.jsl: This file contains the implementation of the CD\_Collection class.
* CDDBIO.jsl: This file contains the implementation of the CDDBIO class.
* Main\_Program.jsl: This file contains the implementation of the main program class.
* AssemblyInfo.jsl: This file is autogenerated by VJ#, and contains information used when building the binary.
* CDDB.txt: This file contains a sample CD database that can be edited with the application.

### Architecture Description

There are four different classes in the CD Library that manage everything from user input to file writing.

* CD: This class encapsulates the information for a single CD, such as Artist and Title. It also has a method to print the information.
* CD\_Collection: This class encapsulates a collection of CDs, and has methods for adding, removing, and displaying CDs, along with some methods for changing the collection’s information.
* CDDBIO: This class encapsulates the file input and output. It has methods to import and export CD collections from and to files.
* Main\_Program: This class is the main class that contains the entry point. It also contains an implementation of a menu system for user input and output.

## Whack-a-PM

Whack-a-PM is a 2d animated Windows game. In it, the player tries to hit a certain PM in the head with a rubber ducky.

### Core Concepts Demonstrated

This sample demonstrates many core concepts that are probably going to be used heavily in the Wumpus game:

* Event handling: Whack-a-PM uses events extensively in the course of a game.
  + Keyboard input
  + Mouse input
  + Timers (as used for animation)
  + Menu options
* Graphics manipulation using GDI+
  + Using bitmap sprite resources to draw game objects
  + Using text output to give the user information
* Embedded Resource manipulation

This sample also demonstrates some useful game concepts which could be helpful or could increase the challenge level:

* Game object behavior encapsulation (Each game object handles its own behavior)
* Inheritance
* Simple audio using DirectSound (This would provide a small challenge for a student)
* Animation (Adding animation would considerably increase the challenge)

### Source File Enumeration

The Whack-a-PM sample has five actual source code files, with a number of supporting files. As usual, each source file typically contains code for a single class.

* DuckObject.jsl, Game.jsl, GameObject.jsl, PMObject.jsl: These each contain source code for the like-named class.
* wapmForm.jsl: This file contains the source for the form. It is a design file and contains generated code for the main window, along with event handling code. This file is broken up into a few separate regions (using the #region tag); the regions can be collapsed in the VJ# IDE to simplify the structure.
* AssemblyInfo.jsl: This file again contains generated information used when building the binary.
* Duck.bmp, PM.bmp, Splash.bmp, tada.wav: These files contain the game resources. These are included as embedded resources (Splash.bmp is also the default background of the main game window).
* wapmForm.resx: Another generated file.

### Architecture Description

Whack-a-PM has five different classes:

* GameObject: This is an abstract class that controls how any game object is drawn. It associates a sprite (from an embedded bitmap) with a game object, and provides methods for changing the location of the object and for drawing the object. Each subclass must override the abstract method UpdateObject. This method is called on each game object for each frame of animation, and each object changes its position accordingly (duckies will accelerate toward the bottom of the screen, and PMs will bob up then down). UpdateObject returns true if the object is still in play, and false if the object has no more action to take. *This is not the same as the Wumpus GameControl object.*
* DuckObject: This class implements the behavior of a ducky. It has a velocity and acceleration associated with it; each time UpdateObject is called, the velocity is changed and the position is changed. When the ducky goes all the way past the bottom edge of the screen, UpdateObject returns false.
* PMObject: This class implements the behavior of a PM. The PM starts offscreen at the bottom and goes up for a while and then goes back down. When the PM gets all the way offscreen UpdateObject returns false.
* wapmForm: This class implements the main game screen. As part of this, it handles all of the user input events for the game (menus, keyboard, mouse). For most inputs, it passes the input directly to the Game object without any processing.
* Game: This class implements the main animation loop. It does the job of loading bitmaps for the GameObject sprites, creating a timer for the animation, pausing, starting and stopping, resetting, and turning sound on and off. When the mouse is clicked, the Game class creates a ducky. When the timer event is triggered, the Game class updates to the next frame of animation (randomly spawn a PM, update the scene by calling UpdateObject on all the players, check for any collisions between a ducky and a PM, and then update the graphics). Updating the graphics involves drawing all the players (duckies and PMs) and then, if necessary, drawing the text for the score. *This class is much like the Game Control object for Wumpus (except that Wumpus won’t need animation).*

## Test Application

The Test Application sample provides buttons for testing each of the methods of the Duck object from the Whack-a-PM sample.

### Core Concepts Demonstrated

This sample demonstrates the proper construction of a test app, which the students will be creating around their specific objects.

* Use a button for each method of the class under test.
* Use a dialog to let the tester determine any parameters for a method. This dialog should pop up when the tester hits a button for one of the methods (unless of course the method doesn’t take any parameters).
* Use a dialog to display any results from a method (that is, if the method returns a string, display that string after calling the method).

In some cases, the students will put the input text boxes for parameters directly on the main form. As long as these are clearly marked, this should cause no problems.

*If the object under test takes keyboard input, the test app should have no buttons or other controls (these will steal the focus and not give it back to the main window).*

### Source File Enumeration/Architecture Description

The Test Application sample contains a good number of designer-generated form source files, along with the necessary files from the Whack-a-PM sample:

* TestApp.jsl: The main test application window. This window contains one button for each of the methods of the DuckObject.
* FormConstructor.jsl: This window is displayed if the Constructor button is clicked on the main window. It contains code for displaying an Open File common dialog, and contains some fields for inputting numbers.
* FormSetLocation.jsl: This window takes input and based on a constructor value parses the input as float or integer. It then calls the DuckObject’s SetLocation.
* FormBoundingRect.jsl: This window is displayed when the DuckObject’s GetBoundingRect is called. It simply displays the return value.
* GameObject.jsl, DuckObject.jsl, Duck.bmp: These are pulled directly from the Whack-a-PM sample, with very small changes to get them building in this sample.
* AssemblyInfo.jsl: The generated file containing build information.
* \*.resx: Resource files for the forms.

# J# Crash Course

## Useful References

For J# syntax, any good Java 2 book will do. Here are some that we’ve used:

* Java 2: The Complete Reference, Fifth Edition by Herbert Schildt.
* Java in a Nutshell by David Flanagan

For the .NET framework, MSDN is a useful reference. The Wumpus alias is also a very good place to go for J# help (we’ve been through it all ourselves).

## Overview of Language Differences

J# is not a widely known language. However, the similarities between J# and a number of other languages make picking up J# very easy. Here are some comparisons between J# and some widely known languages.

### J# versus Java

J# is syntactically identical to Java, with the exception of a few keywords (such as package and #region). This is important because the AP system for Computer Science uses Java as the standard language for tests, and all the syntax learned for the AP test is the same as the syntax used in J#. The difference between J# and Java lies in the libraries that each has available. Java has a standard set of libraries available, along with some add-ons (such as Swing); these libraries are of good quality, but they don’t take advantage of the large amount of work that went into the .NET framework. J# does not have support for most of the Java libraries, but it provides full support for all the .NET libraries available.

Even with the great similarity, after compilation of any sort the results from Java and J# are completely incompatible. For this reason, J# uses a different extension (.jsl) for source code files.

### J# versus C#

J# is similar to C# (about as similar as Java is to C#). One of the big differences is the support for properties. C# has support for class properties that can be accessed like member variables but that can have code behind the scenes for allowing dynamic changes. This is a heavily used feature in the .NET framework, so J# does support this. However, in J# these properties are accessed through methods.

For example, the .NET Image object has public property Flags. In C#, this can be accessed as var = object.Flags; and object.Flags = var. In J#, this property is accessed as var = object.get\_Flags(); and object.set\_Flags(var);.

### J# versus C++

J# is to C++ as Java is to C++ and C# is to C++.

* All code in J# must be in a class (for example, instead of main sitting all by itself, main is a static member of one of the classes in an application).
* Declaration happens at the same time as definition. There are no headers, and there are no prototypes.
* There is no multiple inheritance.
  + An object that inherits from another object is a subclass, while the object inherited from is the superclass.
  + In a subclass’s constructor, super(…) calls the superclass’s constructor.
* Garbage collection happens automatically (no more need to call delete or Release). **Note: in some cases in J#, the .NET Dispose method must be called. The GDI+ Graphics object’s Dispose method Must be called after the object is used.**
* Every variable is a reference to an object (except for some types like int).
  + Corollary: when creating an object, new is used.

# Lectures

The PowerPoint slides are designed to provide you with everything you need to be able to deliver an effective lecture; regardless of whether you present from the slides directly, or rather just use them as private notes to refer to while delivering the lecture.

The following pages contain the lecture slides for all lectures.

# Wumpus Training

These are the slides that will be presented during the Wumpus Training event. These should come in handy as a reference during the project.