

1910716 - Advanced Software Development Assignment 1

by Ollie Bird

Submission date: 10-Dec-2021 05:27PM (UTC+0000)

Submission ID: 165815668

File name: 18246_Ollie_Bird_1910716_-
_Advanced_Software_Development_Assignment_1_382345_113156524.docx (782.63K)

Word count: 1964

Character count: 10202

An Evaluation of the C++ Console Game ‘Adventure Game’ and its Implementation as a Display of Object-Oriented Programming.

Oliver Bird

Apprentice Software Engineer – BT , Data Services

1910716@student.uwtsd.ac.uk

ABSTRACT – A detailed evaluation of the console game ‘Adventure game’ in which its implementation as an object-oriented project will be considered. The game is tested with implementations of happy path end to end testing methods, along with some integration and unit tests. The game is found to be functional and repeatable with randomised values with each playthrough. Consideration is also given to object-oriented programming and its importance in creating such an application.

1. Evaluation

A) C++ has been used to create a simple text-based adventure game allowing the players to fight increasingly difficult enemies in a turn-based format. As an object-oriented (“OO”) language, this has allowed for the use of specific custom object types to allow the game to function in the form of classes.

The use of classes has massively reduced the overall number of lines of code within the application, as there is no need to duplicate object types with different values against their properties. Standardisation of types of objects across the game has meant the central game file has no need for consideration of the objects it instantiates; All methods and properties are contained and managed at the class level.

As seen in Fig. 1 and Fig.2 players can be created easily with custom names and randomised values.

```
Player createPlayer(string name) {
    int health = randomNumber(25, 30) * 4;
    int attackPower = randomNumber(5, 10);
    Player player = Player(name, health, attackPower, Spell(), Weapon());
    player.SetSpell(Spell("Frostbolt", 20));
    player.PrintCharacter();
    player.GetWeapon().printName();
    player.GetWeapon().printAp();
    consoleBreak();
}
```

Fig. 1 - Player instantiation

```
vector<Player> allPlayers;
for (string player : playerNames) {
    // need to add a sleep here to stop the random
    Sleep(1000);
    allPlayers.push_back(createPlayer(player));
}
```

Fig. 2 Looped Player Instantiation

B) Memory referencing has been used within the game to directly access created objects inside of iterator loops. In the specific case detailed below, the ‘&’ is used as a reference when passed to the method to allow the player objects to be modified over the loop.

```
for (Player &player : allPlayers) {
```

Fig. 3 Player Turn Loop

C) Within OO programming, classes may be defined in a ‘parent – child’ relationship [1], allowing for the use of inheritance. The child class may be instantiated with its own unique properties but is able to inherit extended properties and methods from the parent which it extends. Fig. 4, 5 and 6 illustrate this concept within the game via both the player and monster classes. These classes inherit properties from a parent class ‘Character’. This allows common attributes such as *name* and *health* to be set in the parent, while the differences between the classes are assigned in each of them separately; namely that a monster has an attribute of the loot which it can drop, and a player has their weapon.

```

Monster::Monster(string name, int health, int attackPower, Spell spells, vector<Weapon> loot)
: Character(name, health, attackPower, spells)
{
    _loot = loot;
}

```

Fig. 4 Inheriting Monster Constructor

```

Player::Player(string name, int health, int attackPower, Spell spells, Weapon weapon)
: Character(name, health, attackPower, spells)
{
    _weapon = weapon;
}

```

Fig. 5 Inheriting Player Constructor

```

Character::Character(string name, int health, int attackPower, Spell spells)
{
    _name = name;
    _health = health;
    _attackPower = attackPower;
    _spells = spells;
}

```

Fig. 6 Character Specified Constructor 1

D) As seen in Fig.6, objects can be instantiated with no default types. Both *Monster* and *Player* are able to be created whilst dependent on further classes, thus allowing more complex child classes to be constructed, without directly relying on inheritance.

E) Compile Time Polymorphism (CTP) via function overloading has been used to reduce the overall number of methods within the primary source file. CTP has allowed for the same named method to be re-written with different parameter types, drastically improving the readability of the program. In addition, CTP adds no overhead or increase in execution time of the application, as the methods are evaluated at compile time [2]. Two examples in Fig.7 and Fig.8 show this.

```

// <summary>
// Prints out details of the player to the console
// </summary>
// <param name="input">
// The player whos information will be printed
// </param>
void displayCharacterInformation(Player input) {
    input.PrintCharacter();
    Weapon currentWeapon = input.GetWeapon();
    currentWeapon.PrintName();
    currentWeapon.PrintAp();
}

// <summary>
// Prints monster stats to the console
// </summary>
// <param name="input">
// The monster whos stats will be printed
// </param>
void displayCharacterInformation(Monster *input) {
    consoleBreak();
    input->PrintCharacter();
    vector<Weapon> loot = input->GetLoot();
    std::cout << "Loot information for Monster: " << input->GetName() << endl;
    for (Weapon i : loot) {
        i.PrintName();
        i.PrintAp();
    }
    consoleBreak();
}

```

Fig. 7 Display Character Overloading

```

// <summary>
// Attack method for a player with a given attack choice
// </summary>
// <param name="player">
// The player attacking
// </param>
// <param name="choice">
// The attack type chosen by the player
// </param>
// <returns>
// The damage the players attack will deal
// </returns>
int Attack(Player player, int choice) {
    int damageDone;
    if (choice == 1) {
        std::cout << "... " << endl;
        damageDone = player.GetAttack() + player.GetWeapon().GetAp();
        std::cout << player.GetName() << " swings their weapon at the monster! " << endl;
    }
    else {
        std::cout << "... " << endl;
        damageDone = player.GetSpells().GetDamage();
        std::cout << player.GetName() << " casts " << player.GetSpells().GetName() << endl;
    }
    return damageDone;
}

// <summary>
// Attack method for a monster attacking a given player
// </summary>
// <param name="monster">
// The attacking monster
// </param>
// <param name="player">
// The player being attacked
// </param>
// <returns>
// The damage the monster will deal
// </returns>
int Attack(Monster monster, Player player) {
    string target = player.GetName();
    int attackChoice = randomNumber(1, 2);
    int damage;
    if (attackChoice == 1) {
        damage = monster.GetAttack();
        std::cout << "The monster attacks " << target << endl;
    }
    else {
        damage = monster.GetSpells().GetDamage();
        std::cout << "The monster attacks " << target << " with " <<
            monster.GetSpells().GetName() << endl;
    }
    return damage;
}

```

Fig. 8 Attack Method Overloading

F) Runtime polymorphism (RTP) via function overriding has also been implemented via inheritance and virtualisation of parent / base methods. Within the application, two scoreboard types have been implemented in the form of a base generic scoreboard, with a child for to display information regarding a specific player. Base *Scoreboard* methods have been virtualised to allow the child *PlayerScoreboard* class to override them to print information specific to the instantiated player. Similar to the @Override notation for methods in Java, RTP and method overriding in C++ allows a subclass to override the methods provided in the base class, in order to change functionality or to add flexibility to the method and allows many more implementations of it. Figs. 9 and 10 display examples from within AdventureGame.

```

class Scoreboard
{
private:
    int _damageDealt;
    int _monstersKilled;
    int _spellsCast;
    int _lootItems;
    int _damageTaken;
    int _timesHealed;

public:
    Scoreboard();
    void addDamageDealt(int damage);
    virtual void printDamageDealt();
    void addMonstersKilled(int monsters);
    virtual void printMonstersKilled();
    void addSpellsCast(int casts);
    virtual void printSpellsCast();
    virtual void printScoreboard();
    void addLootItemsCount(int loot);
    virtual void printLootItemsCount();
    void addDamageTaken(int damageTaken);
    virtual void printDamageTaken();
    void addTimesHealed();
    virtual void printTimesHealed();
};

```

Fig. 9 Base class method virtualisation

```

class PlayerScoreboard :
    public Scoreboard
{
public:
    PlayerScoreboard();
    PlayerScoreboard(string playerName);
    void printDamageDealt();
    void printMonstersKilled();
    void printSpellsCast();
    void setPlayerName(string name);
    string getPlayerName();
    void printScoreboard();
    void addDamageDealt(int damage);
    void addMonstersKilled(int monsters);
    void addSpellsCast(int casts);
private:
    string _playerName;
    int _damageDealt;
    int _monstersKilled;
    int _spellsCast;
};

```

Fig. 10 Child class method overriding 1

G) Scalability is easily achieved within OOP due to simple instantiation of objects. In the case of *AdventureGame*, object creation has been limited due to the nature of the game. Players are limited to 4, and monsters to 5. This is to purely for demonstrative purposes and could be scaled much larger in later versions. With the use of classes, it is simple to make changes to the underlying design of all aspects of the game, potentially such as adding new spells, adding further combat choices such as healing or using items within an inventory system. The *player* class could be modified to allow for these kinds of additions, and only the instantiation method calls would need to be modified. Additional logic could be added for these; however the game would still function in its current state regardless of these additions to the described classes.

H) Security is considered minimally within the *AdventureGame* application. Implementations of private attributes, string management and validation and specific method input types are used throughout the application, to provide stability and security. Coding standards such as ISO 14882:2020 [3] or SEI CERT C++ [4] have not been considered.

I) OOP allows for easier code readability and partitioning into smaller 'chunks' when debugging. The use of classes, class methods and inheritance allow for code to be split into its relevant and related parts easily. Methods are more easily identifiable and understandable from their creation, as object types and descriptions are obvious in both their calling and display. Readability is demonstrated in Figs. 11 and 12.

```

bool arePlayersDead(vector<Player> players) {
    int numberPlayersDead = 0;
    for (Player player : players) {
        if (player.GetHealth() < 1) {
            numberPlayersDead++;
        }
    }
    if (numberPlayersDead == players.size()) {
        return true;
    }
    return false;
}

```

Fig. 11 Readable method declaration

Fig.11 shows easily its function, only by reading the method name and observing its input types. Without reading the underlying logic, it is easy to discern that the method will return a Boolean depending if all players are alive or dead.

```

monster.GetHealth()

```

Fig. 12 Readable method call

The natural separation of object types also allows for clearer testing strategies. Unit tests for each method and class reduce the need for their testing within the scope of where they are instantiated. Developers can have more confidence in their code due to this.

2. Testing

Testing strategies will be employed as shown in Table 1. Both end to end tests and unit tests [5] will be employed to evaluate the functionality of the code. Some code has been left within the application purely to demonstrate testing of custom object types.

Table 1

Test Name + Type	Test Description	Expected outcome + status
End to end happy path – E2E integration	A full test of the game with 1 player and 2 monsters	The game will play to completion and quit upon finish – PASSED
Input Sanitisation - unit	Incorrect input types passed to <i>getline</i> methods	Input methods should not allow incorrect input types – PASSED
Player death - Integration	A player will be killed by a monster	Game of 2 players will continue with 1 player dead – PASSED
All player death - Integration	All players will be killed	Game of 2 players will end when all players are dead – PASSED
Pickup loot - Unit	Attempt to pickup loot from a slain monster	The associated loot object will be returned from the <i>pickLoot</i> method – PASSED
Reject loot pickup - Unit	Choose to keep the current equipped weapon	The weapon will not be updated – PASSED
Weapon pickup updates the attack - Integration	Killing a monster and looting a new weapon and attacking the next monster	The weapon attack damage will be correctly passed upon looting – PASSED
Player heals - Integration	Test if a player is healed between fights	The player should be healed for 50 health if below 100 between fights – PASSED

Game restart - Unit	Attempt to restart the game upon completion	The game should restart if that option is selected – PASSED
Display Scoreboard - Unit	Request to display the scoreboard upon completion	The scoreboard will display – PASSED
Game quit - Unit	Attempt to quit the game upon completion	The game will close – PASSED
Display player scoreboard individually - Integration	Attempt to load the player scoreboard	The scoreboard will load and populate – FAILED

A) End to end happy path – E2E Integration

The game is initialised with 1 player and 2 monsters.

```

Welcome to your adventure!!
-----
<<<< Adventure Game Rules + Explanations! >>>>
1. Please only input values presented by the game. The game will not accept incorrect types or options.
2. Players cannot be resurrected upon death. They will remain out of the game until the end.
3. Players will periodically heal between battles if they are low health.
4. Loot will be assigned to the player who deals the killing blow. Make sure to consider this when playing in groups!
5. The aim of the game is to slay all the monsters, whilst taking the least damage from them as a group.
6. Game details can be displayed after the game is completed.
-----
Would you like to play (y/n) ?
  
```

```

How many Players would you like ? Please limit Players to 4
-----
PLEASE INSERT THIS AS A NUMBER!!
1
You have selected 1 Players
How many Monsters would you like ? Please limit Monsters to 5
-----
PLEASE INSERT THIS AS A NUMBER!!
2
You have selected 2 Monsters
--- Character Name: Player 1
Character Attack: 9
Character Health: 116
Spell Name: Frostbolt
Spell Damage: 20
Weapon Name: fists
Weapon Attack: 1
-----
Are you ready...?
Loading -----
  
```

```

The monster has: 20 health
Player 1, do you wish to:
1: Attack with your weapon for approximately: 5 + 1 damage
2: Use: Frostbolt for 20 damage
2
...
Player 1 casts Frostbolt
player deals: 20
-----
The monster takes: 20 damage!
-----
-----
Player 1 has slain the monster!!!
-----
  
```

```

Please choose the number of the weapon you wish to loot!
0- Rat Tail Bow
1- Foam Sword
2- Lion Claw Dagger
3- Rat Tail Bow
4 keep my current weapon!!!
Please only input the number of the weapon you wish to loot!
3
Player 1 has equiped Rat Tail Bow
-----
Player 1 has been healed for 50 health!
-----
A Skeletal Vulture emerges from the mist...
The monster is powerful and has the following stats:
--- Character Name: Skeletal Vulture
Character Attack: 30
Character Health: 200
Spell Name: Lightning bolt
Spell Damage: 18
-----
The monster has: 200 health
Player 1, do you wish to:
1: Attack with your weapon for approximately: 9 + 60 damage
2: Use: Frostbolt for 20 damage
1
...
Player 1 swings their weapon at the monster!
Player deals: 69
-----
The monster takes: 69 damage!
-----
The monster takes: 69 damage!
-----
Player 1 has slain the monster!!!
-----
The Skeletal Vulture falls to the ground with a thud!
A small satchel appears underneath
The bag falls open showing its contents!
-----
Weapon Name: Sword of a thousand truths
Weapon Attack: 100
-----
Weapon Name: Spear of Destiny
Weapon Attack: 50
-----
Weapon Name: Bone knife
Weapon Attack: 20
-----
Weapon Name: Sword of a thousand truths
Weapon Attack: 100
-----
Silence echoes over the land
-----
The game has ended
All monsters have been slain, loot has been earned and players may now rest
----- Game Stats -----
Total Damage for all players: 307
Number of monsters killed for all players: 2
Total Spells Cast for all players: 5
Total items looted for all players: 2
Total damage taken by all players : 104
Number of times healed for all players: 1
-----
Thankyou for playing
-----
<<<<<<< Menu >>>>>>>
Would you like to:
1- Play Again
2- See the scoreboard
3- Quit the game

```

The game successfully completes once all monsters are slain.

B) Input Sanitisation – Unit

Incorrect input types will be passed to input methods to test if they are accepted.

```

How many Players would you like ? Please limit Players to 4
-----
PLEASE INSERT THIS AS A NUMBER!!
fifty
Please input the number of players as an integer, not a string!
10
You have entered too many Players !
Please add fewer!
4
You have selected 4 Players
How many Monsters would you like ? Please limit Monsters to 5
-----
PLEASE INSERT THIS AS A NUMBER!!
one hundred
Please input the number of players as an integer, not a string!
12
You have entered too many Monsters !
Please add fewer!
5
You have selected 5 Monsters

```

As shown, incorrect values are not accepted

C) Player death – Integration

The game will be run until the player dies. The game should finish as there is only a single player present.

```

<<<<<<<< Monster Turn! >>>>>>>>>
-----
Player 1's current health is: 16
The monster attacks Player 1 with Death Slash
Player 1 takes: 16 damage!
Player 1's new health is: 0
Alas ... Player 1 has fallen...All players are dead
The game will now end :(
-----
Saving in progress
Please wait
-----
Silence echoes over the land
-----
The game has ended
Unfortunately you were unable to slay all the monsters.
----- Game Stats -----
Total Damage for all players: 56
Number of monsters killed for all players: 0
Total Spells Cast for all players: 0
Total items looted for all players: 0
Total damage taken by all players : 108
Number of times healed for all players: 0
-----
Thankyou for playing
-----
<<<<<<< Menu >>>>>>>
Would you like to:
1- Play Again
2- See the scoreboard
3- Quit the game

```

The game ends successfully upon single player death

D) All player death – Integration

The game should end upon all players being slain

```
The monster has: 170 health
Player 1, do you wish to:
1: Attack with your weapon for approximately: 8 + 1 damage
2: Use: Frostbolt for 20 damage
1
...
Player 1 swings their weapon at the monster!
player deals: 9
-----
The monster takes: 9 damage!
-----
<<<<<<<< Monster Turn! >>>>>>>>>
-----
Player 1's current health is: 6
The monster attacks Player 1
Player 1 takes: 30 damage!
Player 1's new health is: -24
Alas ... Player 1 has fallen...
The monster has: 161 health
Player 2, do you wish to:
1: Attack with your weapon for approximately: 5 + 1 damage
2: Use: Frostbolt for 20 damage
```

Player 2 can continue playing whilst player 1 is slain.

```
...
Player 2 swings their weapon at the monster!
player deals: 6
-----
The monster takes: 6 damage!
-----
<<<<<<<< Monster Turn! >>>>>>>>>
-----
Player 2's current health is: 25
The monster attacks Player 2
Player 2 takes: 30 damage!
Player 2's new health is: -5
Alas ... Player 2 has fallen...All players are dead
The game will now end :(
-----
Saving in progress
Please wait
-----
```

The game completes successfully once all players have been slain.

E) Pickup loot – Unit

A player should be able to equip a new weapon upon killing a monster.

```
Please choose the number of the weapon you wish to loot!
0- Lion Claw Dagger
1- Rat Tail Bow
2- Foam Sword
3- Lion Claw Dagger
- 4 keep my current weapon!!!
Please only input the number of the weapon you wish to loot!
3
Player 1 has equiped Lion Claw Dagger
-----
Player 1 has been healed for 50 health!
-----
A Burning Wookie emerges from the mist...
The monster is powerful and has the following stats:
--- Character Name: Burning Wookie
Character Attack: 30
Character Health: 200
Spell Name: Lightning bolt
Spell Damage: 26
-----
The monster has: 200 health
Player 1, do you wish to:
1: Attack with your weapon for approximately: 8 + 55 damage
2: Use: Frostbolt for 20 damage
```

The player has successfully equipped a new weapon.

F) Reject loot pickup – Unit

A player should be able reject a weapon pickup if their weapon is more desirable.

```
Please choose the number of the weapon you wish to loot!
0- Spear of Destiny
1- Bone knife
2- Foam Sword
3- Spear of Destiny
- 4 keep my current weapon!!!
Please only input the number of the weapon you wish to loot!
4
Player 1 has equiped Lion Claw Dagger
```

The player was able to keep their current weapon and continue to play the game.

G) Weapon pickup updates the attack – Integration

The players overall attack should be updated after picking up a new weapon.

```
Player 1, do you wish to:
1: Attack with your weapon for approximately: 8 + 1 damage
2: Use: Frostbolt for 20 damage
```

The player has his original attack prior to slaying the monster and looting a new weapon.

```

Please choose the number of the weapon you wish to loot!
0- Lion Claw Dagger
1- Rat Tail Bow
2- Foam Sword
3- Lion Claw Dagger
4- 4 keep my current weapon!!!
Please only input the number of the weapon you wish to loot!
3
Player 1 has equipped Lion Claw Dagger
-----
Player 1 has been healed for 50 health!
-----
A Burning Wookie emerges from the mist...
The monster is powerful and has the following stats:
--- Character Name: Burning Wookie
Character Attack: 30
Character Health: 200
Spell Name: Lightning bolt
Spell Damage: 26
-----
The monster has: 200 health
Player 1, do you wish to:
1: Attack with your weapon for approximately: 8 + 55 damage

```

The player's attack has been updated with the damage value corresponding to the new weapons attack value.

H) Player heals between turns - Integration

The player should heal for 50 health between each battle, given that they were below 50 upon battle completion.

```

-----
<<<<<<<< Monster Turn! >>>>>>>>>
-----
Player 1's current health is: 71
The monster attacks Player 1
Player 1 takes: 15 damage!
Player 1's new health is: 56

```

The player is on 56 health prior to slaying the monster

```

Please choose the number of the weapon you wish to loot!
0- Lion Claw Dagger
1- Rat Tail Bow
2- Foam Sword
3- Lion Claw Dagger
4- 4 keep my current weapon!!!
Please only input the number of the weapon you wish to loot!
3
Player 1 has equipped Lion Claw Dagger
-----
Player 1 has been healed for 50 health!
-----

```

```

-----
<<<<<<<< Monster Turn! >>>>>>>>>
-----
Player 1's current health is: 106
The monster attacks Player 1
Player 1 takes: 30 damage!
Player 1's new health is: 76

```

The player is successfully healed as shown by the health values.

I) Game restart upon completion - Unit

The game should be able to be restarted upon selecting the correct option in the ending menu.

```

<<<<<<<< Menu >>>>>>>>>
Would you like to:
1- Play Again
2- See the scoreboard
3- Quit the game

```

```

Welcome to your adventure!!
-----
<<< Adventure Game Rules & Explanations! >>>
1. Please only input values requested by the game. The game will not accept incorrect types or options.
2. Players cannot be resurrected upon death. They will remain out of the game until the end.
3. Players will periodically heal between battles if they are low health.
4. Loot will be assigned to the player who deals the killing blow. Make sure to consider this when playing in groups!
5. The aim of the game is to slay all the monsters, whilst taking the least damage from them as a group.
6. Game details can be displayed after the game is completed.
-----
Would you like to play (y/n) ?

```

The game successfully restarts and is playable.

J) Display Scoreboard – Unit

The final scoreboard should display correctly when the play selects the associated value.

```

<<<<<<<< Menu >>>>>>>>>
Would you like to:
1- Play Again
2- See the scoreboard
3- Quit the game
2
----- Game Stats -----
Total Damage for all players: 100
Number of monsters killed for all players: 1
Total Spells Cast for all players: 5
Total items looted for all players: 1
Total damage taken by all players : 69
Number of times healed for all players: 0
The game will begin again in 30 seconds, please close to cancel

```

The final scoreboard displays as expected.

K) Game quits upon completion – Unit

The game should quit upon the exit game option being selected

```

<<<<<<<< Menu >>>>>>>>>
Would you like to:
1- Play Again
2- See the scoreboard
3- Quit the game
3
-----
Thankyou for playing!

```

The game quits as expected.

J) Display player scoreboard individually – Integration

Player scoreboards are able to be instantiated and have access to their inherited attributes and methods from the generic scoreboard class as shown below.

```
vector<PlayerScoreboard> scoreboards;
for (int playerScoreboardNumber = 1; playerScoreboardNumber <= allPlayers.size(); playerScoreboardNumber++) {
    string name = "Player" + playerScoreboardNumber;
    PlayerScoreboard sbName = PlayerScoreboard(name);
    scoreboards.push_back(sbName);
}
```

Binding these scoreboards to a specific player object was unable to be completed, and hence the objects remain empty and do not update for each specific player. A more efficient implementation is needed with considerations for scoreboards being directly linked to players, or via an index based loop as opposed to an iterator. The proposed test will always fail in this case.

3. Video Submission

The following link will provide access to a video demonstration of the game. This will detail 2 playthroughs of the game showing actions for both single and multiplayer.

<https://youtu.be/Xd89VR5SoGQ>

References

- [1] M. Patel, "Inheritance and Its Type in Object Oriented Programming Using C++," *International Journal of Recent Technology and Engineering*, vol. 8, no. 4, pp. 1991-1998, 10 September 2019.
- [2] B. Kahanwal, "Function Overloading Implementation in C++," *International Journal of Advances in Engineering Sciences*, vol. 3, no. 4, pp. 12-18, 2013.
- [3] ISO, "ISO/IEC 14882:2020," 12 2020. [Online]. Available: <https://www.iso.org/standard/79358.html?browse=tc>. [Accessed 01 12 2021].
- [4] "SEI CERT C++ Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems (2016 Edition)," March 2017. [Online]. Available: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=494932>. [Accessed 02 12 2021].
- [5] H. Vocke, "The Practical Test Pyramid," 26 February 2018. [Online]. Available: <https://martinfowler.com/articles/practical-test-pyramid.html>. [Accessed 20 11 2021].

1910716 - Advanced Software Development Assignment 1

GRADEMARK REPORT

FINAL GRADE

93/100

GENERAL COMMENTS

Instructor

Classes (25): 23
Polymorphism (10): 10

Game (20): 18

Report (35): 32

Testing (5): 5

Demo (5): 5

Feedback:
=====

Report could have provided a little more discussion of data encapsulation.

Fairly interesting code with a number of classes provided. The AdventureGame code could be split into some additional classes.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8
