SENG 330
Assignment 2

Building a REAL Project
Using a subset from group assignment

Jacob Smith
V00700979

1 Github
Repository Link:
https://github.com/jwsmithgit/seng330ass2/

2 Installation
INSTALL.txt:

```
Tools Needed:
- Visual Studio

How to Build:
- Open /seng330ass2.vcxproj with Visual Studio
- Build the program in Visual Studio with ctrl+shift+b

How to Run:
- Run the program in Visual Studio with ctrl+f5


To View the Doxygen:
- Open /html/index.html
- Sample pictures are provided in the root folder as doxygenscreen*.png
```

3 Prototype Design Pattern

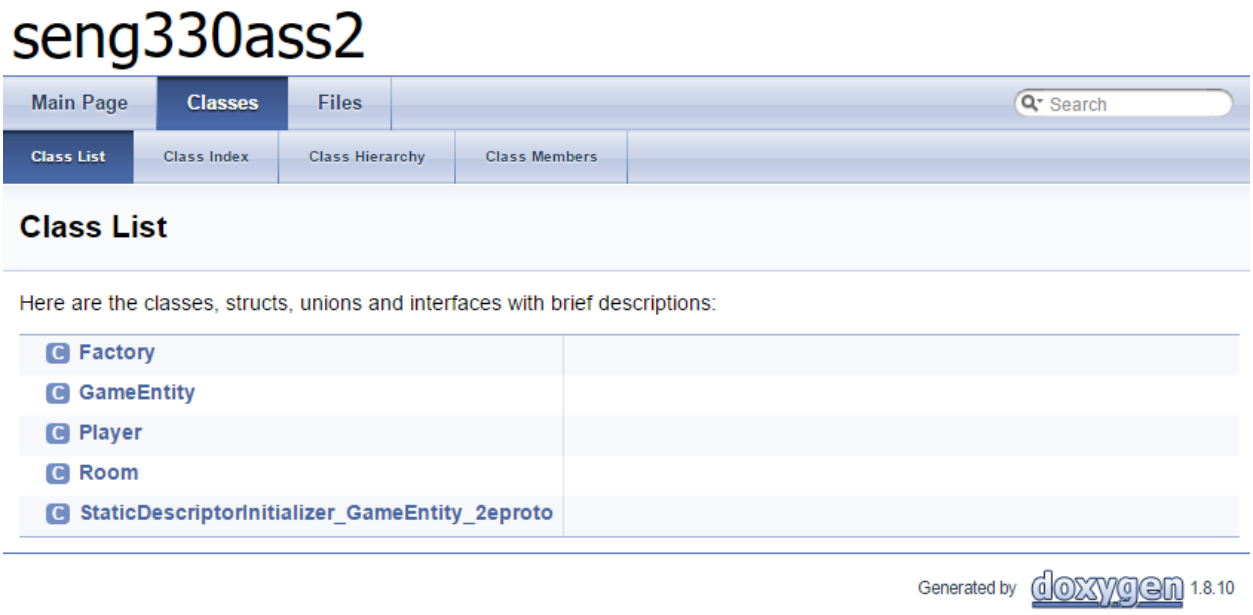
- Subset of group project
- GameEntity is the base class
- Player and Room are the derived classes
- Factory class is implemented to delegate the correct clones.
- Source code is at end of document and on the GIthub

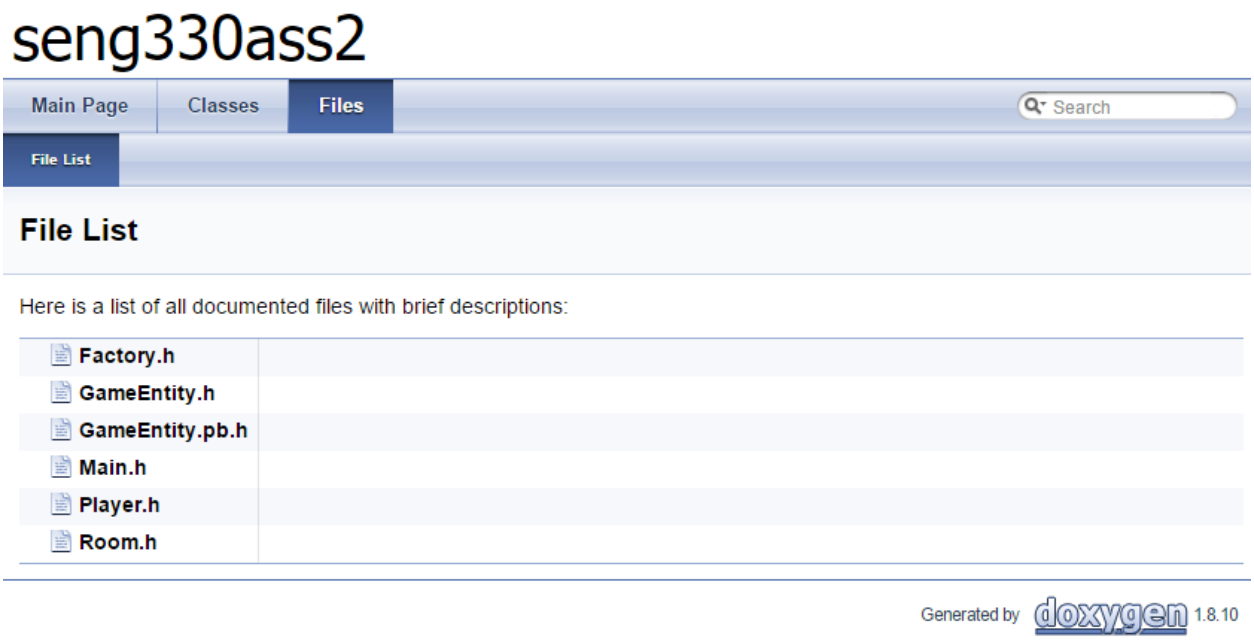
4 Google Protocol Buffers
GameEntity.proto file

```
message GameEntity {
    required int32 id = 1;
    required string name = 2;
    optional string description = 3;
}
```

Unfortunately, I could not get the protocol buffers source to compile with Visual Studio. It produces many errors upon compiling, so I was not able to get the .lib for compile-time linking in my program. The problem appears to be that Visual Studio 2015 does not provide stdext. Which is required since the Google Proto Source uses hash_maps and hash_sets.
Tried many hours to get it to work, but to no avail. Part marks for this one hopefully.

5 Doxygen
Screenshots:





Static Public Member Functions | List of all members

Factory Class Reference

Static Public Member Functions

static **GameEntity** * **MakeGameEntity** (int choice)

Member Function Documentation

GameEntity * **Factory::MakeGameEntity** (**int choice**) static

Clones a copy of object in prototypes array

The documentation for this class was generated from the following files:

- **Factory.h**
- Factory.cpp

Player Class Reference

Inheritance diagram for Player:

Public Member Functions

	Player ()
	Player (int id, std::string name)
	Player (int id, std::string name, std::string description)
	~Player ()
Player *	**Clone** ()

▸ **Public Member Functions inherited from GameEntity**

Additional Inherited Members

▸ **Static Public Member Functions inherited from GameEntity**

▸ **Static Public Attributes inherited from GameEntity**

6 Google Test

Like Google Protocol Buffer, took a long time to get working with Visual Studio. I will not be learning/using Visual Studio in the future, probably will use CMake. No time to go back and change/learn now.

Problem has been solved, the problem was with linking files together, having both the regular code and the test code pointing to a base library code. Learned a lot. At least the codebase compiled with Visual Studio, unlike the Protocol Buffers.

Test Cases:

- Created test cases for a basic factorial method as shown in the learning guide
- Created test case for Player, verifying constructor and getters
- Created test case for Player, verifying setters and getters
- Created test case for Room, verifying constructor and getters
- Created test case for Room, verifying setters and getters

Code:

Factory.h

```
#ifndef __FACTORY_H__
#define __FACTORY_H__

#include "GameEntity.h"
#include "Player.h"
#include "Room.h"

class Factory {
	private:
		static GameEntity* Prototypes[3];
	public:
		static GameEntity* MakeGameEntity(int choice);
};

#endif // __FACTORY_H__
```

Factory.cpp

```
#include "Factory.h"

/**
* Array of prototypes for factory
*/
GameEntity* Factory::Prototypes[] = {
	0, new Player, new Room
};

/**
* Clones a copy of object in prototypes array
*/
GameEntity* Factory::MakeGameEntity(int choice) {
	return Prototypes[choice]->Clone();
}
```

GameEntity.h

```
#ifndef __GAMEENTITY_H__
#define __GAMEENTITY_H__

#include <string>
#include <iostream>

class GameEntity {

private:
    int id_;
    std::string name_;
    std::string description_;

public:
       GameEntity();
    GameEntity(int id, std::string name);
    GameEntity(int id, std::string name, std::string description);
    ~GameEntity();

       virtual GameEntity* Clone() = 0;

       void Print();
       void PrintName();

    int GetId();
       std::string GetName();
    std::string GetDescription();

       void SetId(int id);
       void SetName(std::string name);
    void SetDescription(std::string description);

};

#endif // __GAMEENTITY_H__
```

GameEntity.cpp

```
#include "GameEntity.h"

#include <iostream>

/**
* constructors, one for each possibility of initial variables
*/
GameEntity::GameEntity() {
       id_ = 0;
       name_ = "";
       description_ = "";

       std::cout << "Created an entity..." << std::endl;
}

GameEntity::GameEntity(int id, std::string name) {
    id_ = id;
    name_ = name;
    description_ = "";
```

```
    std::cout << "Created an entity..." << std::endl;
}

GameEntity::GameEntity(int id, std::string name, std::string description) {
    id_ = id;
    name_ = name;
    description_ = description;

    std::cout << "Created an entity..." << std::endl;
}

/**
* destructor
*/
GameEntity::~GameEntity() {
    std::cout << "Destroyed an entity..." << std::endl;
}

/**
* print out information
*/
void GameEntity::Print() {
       std::cout << id_ << " " << name_ << " " << description_ << std::endl;
}

void GameEntity::PrintName() {
       std::cout << name_ << std::endl;
}

/**
* getters
*/
int GameEntity::GetId() {
    return id_;
}

std::string GameEntity::GetName() {
       return name_;
}

std::string GameEntity::GetDescription() {
    return description_;
}

/**
* setters
*/
void GameEntity::SetId(int id) {
       id_ = id;
}

void GameEntity::SetName(std::string name) {
       name_ = name;
}

void GameEntity::SetDescription(std::string description) {
    description_ = description;
}
```

Player.h

```h
#ifndef __PLAYER_H__
#define __PLAYER_H__

#include "GameEntity.h"

class Player: public GameEntity{

private:

public:
	Player();
    Player(int id, std::string name);
	Player(int id, std::string name, std::string description);
    ~Player();

	Player* Clone();

};

#endif
```

Player.cpp

```cpp
#include "Player.h"

/**
* Player Constructors, use GameEntity constructor
*/
Player::Player() : GameEntity() {
	std::cout << "Created a player..." << std::endl;
}

Player::Player(int id, std::string name) : GameEntity(id, name){
    std::cout << "Created a player..." << std::endl;
}

Player::Player(int id, std::string name, std::string description) : GameEntity(id,
name, description) {
	std::cout << "Created a player..." << std::endl;
}

/**
* Player Deconstructor
*/
Player::~Player() {
    std::cout << "Destroyed a player..." << std::endl;
}

/**
* Clones a player, returns empty object
*/
Player* Player::Clone() {
	std::cout << "Cloned a player.." << std::endl;
	return new Player;
}
```

Room.h

```h
#ifndef __ROOM_H__
#define __ROOM_H__

#include "GameEntity.h"

class Room : public GameEntity {

private:

public:
	Room();
	Room(int id, std::string description);
	Room(int id, std::string name, std::string description);
	~Room();

	Room* Clone();

};

#endif // __ROOM_H__
```

Room.cpp

```cpp
#include "Room.h"

/**
* Room Constructors, use GameEntity constructor
*/
Room::Room() : GameEntity() {
	std::cout << "Created a room..." << std::endl;
}

Room::Room(int id, std::string name) : GameEntity(id, name) {
	std::cout << "Created a room..." << std::endl;
}

Room::Room(int id, std::string name, std::string description) : GameEntity(id, name,
description) {
	std::cout << "Created a room..." << std::endl;
}

/**
* Room Deconstructor
*/
Room::~Room() {
	std::cout << "Destroyed a room..." << std::endl;
}

/**
* Clones a room, returns empty object
*/
Room* Room::Clone() {
	std::cout << "Cloned a room..." << std::endl;
	return new Room;
}
```

Main.cpp

```cpp
/**
* Jacob Smith
* V00700979
* Seng 330
* Assignment 2
*/

/**
* included files
*/
#include "GameEntity.h"
#include "Factory.h"

#include <iostream>
#include <vector>

//#include "GameEntity.pb.h"
//#include <fstream>

int main(int argc, char* argv[]){

	/**
	* setup variables
	*/
	std::vector<GameEntity*> entities;
	int choice;
	std::string name;

	/**
	* get user input and create clones
	*/
	while (true) {
		std::cout << "\nChoose GameEntity type to create, Player(1) Room(2) Go(0): ";
		std::cin >> choice;
		if (choice == 0)
			break;

		std::cout << "Enter name of GameEntity: ";
		std::getline(std::cin, name); // clears cin from int grabbing above
		std::getline(std::cin, name);
		//std::cin >> name;

		GameEntity* Entity = Factory::MakeGameEntity(choice);
		Entity->SetName(name);

		entities.push_back(Entity);
	}

	/**
	* print out all clones created before exiting
	*/
	std::cout << "Objects created: ";
	for (int i = 0; i < entities.size(); ++i)
		entities[i]->PrintName();
```

```
	/**
	* google protocol buffer output, not working, commented out
	*/
	//std::fstream output("myfile", std::ios::out | std::ios::binary);
	//for (int i = 0; i < entities.size(); ++i)
		//entities[i]->SerializeToOstream(&output);

	/**
	* delete all the created clones
	*/
	for (int i = 0; i < entities.size(); ++i)
		delete entities[i];

	/**
	* allow user to see output before exiting
	*/
	std::cout << "Press enter to exit...";
	std::cin.get();
	std::cin.get();
}
```

Test.cpp

```
#include "gtest\gtest.h"
#include "..\player.h"
#include "..\room.h"

#include <string>

using std::string;

int Factorial(int n) {
	if (n <= 0)
		return 1;
	return n * Factorial(n - 1);
}

TEST(FactorialTest, HandlesZeroInput) {
	EXPECT_EQ(1, Factorial(0));
}


TEST(FactorialTest, HandlesPositiveInput) {
	EXPECT_EQ(1, Factorial(1));
	EXPECT_EQ(2, Factorial(2));
	EXPECT_EQ(6, Factorial(3));
	EXPECT_EQ(40320, Factorial(8));
}

TEST(FactorialTest, HandlesNegativeInput) {
	EXPECT_EQ(1, Factorial(-5));
	EXPECT_EQ(1, Factorial(-1));
	EXPECT_GT(Factorial(-10), 0);
}

class PlayerTest : public ::testing::Test {
```

```
	protected:
		virtual void SetUp() {
			player = new Player(0, "Greg", "Has name Greg");
		}
		Player* player;
};

TEST_F(PlayerTest, GregIsReal) {
	EXPECT_EQ(0, player->GetId());
	EXPECT_STREQ("Greg", player->GetName().c_str());
	EXPECT_STREQ("Has name Greg", player->GetDescription().c_str());
}

TEST_F(PlayerTest, GregChangeToLana) {
	player->SetId(1);
	player->SetName("Lana");
	player->SetDescription("Has name Lana");

	EXPECT_EQ(1, player->GetId());
	EXPECT_STREQ("Lana", player->GetName().c_str());
	EXPECT_STREQ("Has name Lana", player->GetDescription().c_str());
}

class RoomTest : public ::testing::Test {
protected:
	virtual void SetUp() {
		room = new Room(0, "Dungeon", "Looks dark");
	}
	Room* room;
};

TEST_F(RoomTest, DungeonIsReal) {
	EXPECT_EQ(0, room->GetId());
	EXPECT_STREQ("Dungeon", room->GetName().c_str());
	EXPECT_STREQ("Looks dark", room->GetDescription().c_str());
}

TEST_F(RoomTest, DungeonChangeToSunnyField) {
	room->SetId(1);
	room->SetName("Sunny Field");
	room->SetDescription("Looks beautiful!");

	EXPECT_EQ(1, room->GetId());
	EXPECT_STREQ("Sunny Field", room->GetName().c_str());
	EXPECT_STREQ("Looks beautiful!", room->GetDescription().c_str());
}
```