

Project 1: Battle for Zion Testing

There were 16 test cases. Each test was worth 10/16 points; to run the test cases:

1. Remove the main routine from your `main.cpp` file.
2. For each test case, add the appropriate test code to your `main.cpp` file.
3. Re-compile the code and run it in 'g32' on the Linux SEAS UCLA server `lnxsr07.seas.ucla.edu`.

Test Case	Command Line and Code Used for main.cpp
1. Verify compilation of .cpp files in g32 is completed successfully with all previously defined interfaces not changed	<pre> UNIX COMMAND LINE: g32 main.cpp Arena.cpp Game.cpp Player.cpp Robot.cpp Previous.cpp utilities.cpp main.cpp: #include <cassert> #include "Robot.h" #include "Player.h" #include "Arena.h" #include "Game.h" #define CHECKTYPE(f, t) { (void)(t)(f); } void thisFunctionWillNeverBeCalled() { // If the student deleted or changed the interfaces to the public // functions, this won't compile. (This uses magic beyond the scope // of CS 32.) Robot(static_cast<Arena*>(0), 1, 1); CHECKTYPE(&Robot::row, int (Robot::*)() const); CHECKTYPE(&Robot::col, int (Robot::*)() const); CHECKTYPE(&Robot::move, void (Robot::*)()); CHECKTYPE(&Robot::takeDamageAndLive, bool (Robot::*)()); Player(static_cast<Arena*>(0), 1, 1); CHECKTYPE(&Player::row, int (Player::*)() const); CHECKTYPE(&Player::col, int (Player::*)() const); </pre>

```

CHECKTYPE(&Player::age,
int      (Player::*)() const);
CHECKTYPE(&Player::isDead,
bool     (Player::*)() const);
CHECKTYPE(&Player::takeComputerChosenTurn,
string   (Player::*)());
CHECKTYPE(&Player::stand,
void     (Player::*)());
CHECKTYPE(&Player::move,
void     (Player::*)(int));
CHECKTYPE(&Player::shoot,
bool     (Player::*)(int));
CHECKTYPE(&Player::setDead,
void     (Player::*)());

Arena(1, 1);
CHECKTYPE(&Arena::rows,      int
(Arena::*)() const);
CHECKTYPE(&Arena::cols,      int
(Arena::*)() const);
CHECKTYPE(&Arena::player,
Player* (Arena::*)() const);
CHECKTYPE(&Arena::robotCount, int
(Arena::*)() const);
CHECKTYPE(&Arena::nRobotsAt,  int
(Arena::*)(int,int) const);
CHECKTYPE(&Arena::display,    void
(Arena::*)(string) const);
CHECKTYPE(&Arena::addRobot,    bool
(Arena::*)(int,int));
CHECKTYPE(&Arena::addPlayer,   bool
(Arena::*)(int,int));
CHECKTYPE(&Arena::damageRobotAt, void
(Arena::*)(int,int));
CHECKTYPE(&Arena::moveRobots,  bool
(Arena::*)());

Game(1,1,1);
CHECKTYPE(&Game::play, void
(Game::*)());

Previous(1, 1);
CHECKTYPE(&Previous::dropACrumb, bool
(Previous::*)(int,int));

CHECKTYPE(&Previous::showPreviousMoves,
void (Previous::*)() const);

```

	<pre> } int main() { </pre>
2. Verify multiple inclusions of .h file compiles properly	<p>UNIX COMMAND LINE: g32 main.cpp Game.cpp Arena.cpp Previous.cpp Robot.cpp utilities.cpp</p> <p>main.cpp:</p> <pre> #include "Game.h" #include "Game.h" #include "Arena.h" #include "Arena.h" #include "Previous.h" #include "Previous.h" #include "Player.h" #include "Player.h" #include "Robot.h" #include "Robot.h" #include "globals.h" #include "globals.h" int main() { </pre>
3. Verify basic Previous class functions work	<p>UNIX COMMAND LINE: g32 main.cpp Previous.cpp</p> <p>main.cpp:</p> <pre> #include "Previous.h" int main() { Previous p(2, 2); p.dropACrumb(1, 1); p.showPreviousMoves(); } </pre>
4. Verify Robot is created	<p>UNIX COMMAND LINE: g32 main.cpp Robot.cpp</p> <p>main.cpp:</p> <pre> #include "Robot.h" int main() { Robot r(0, 1, 1); } </pre>
5. Verify Player is created	<p>UNIX COMMAND LINE: g32 main.cpp Player.cpp</p> <p>main.cpp:</p> <pre> #include "Player.h" </pre>

	<pre> int main() { Player p(0, 1, 1); } </pre>
6. Verify Player is added to Arena successfully	<p>UNIX COMMAND LINE: g32 main.cpp Arena.cpp</p> <p>main.cpp:</p> <pre> #include "Arena.h" int main() { Arena a(10, 18); a.addPlayer(2, 2); } </pre>
7. Verify Arena and Player initialized successfully	<p>UNIX COMMAND LINE: g32 main.cpp Player.cpp Arena.cpp</p> <p>main.cpp:</p> <pre> #include "globals.h" #include "Player.h" #include "Arena.h" int main() { Arena a(10, 20); Player p(&a, 2, 3); } </pre>
8. Verify Arena and Player initialized successfully if Arena.h included Previous Player.h	<p>UNIX COMMAND LINE: g32 main.cpp Player.cpp Arena.cpp</p> <p>main.cpp:</p> <pre> #include "Arena.h" #include "Player.h" int main() { Arena a(10, 20); Player p(&a, 2, 3); } </pre>
9. Verify Arena and Player initialized successfully if Player.h included Previous Arena.h	<p>UNIX COMMAND LINE: g32 main.cpp Player.cpp Arena.cpp</p> <p>main.cpp:</p> <pre> #include "Player.h" #include "Arena.h" int main() { Arena a(10, 20); Player p(&a, 2, 3); } </pre>

	<pre> } </pre>
10. Verify proper tracking of Player movement after moving	<pre> UNIX COMMAND LINE: g32 main.cpp Arena.cpp Game.cpp Previous.cpp Player.cpp Robot.cpp utilities.cpp main.cpp: #include <string> #include "Arena.h" #include "Player.h" #include "Previous.h" #include "globals.h" #include <sstream> #include <iostream> using namespace std; class StreambufSetter { public: StreambufSetter(ios& str, streambuf* sb) : stream(str), oldsb(str.rdbuf(sb)) {} ~StreambufSetter() { stream.rdbuf(oldsb); } private: ios& stream; streambuf* oldsb; }; int main() { ostringstream oss; StreambufSetter ssout(cout, oss.rdbuf()); Arena a(2, 2); a.addPlayer(1, 1); a.player()->move(RIGHT); a.player()->move(DOWN); a.thePrevious().showPreviousMoves(); string s = oss.str(); string::size_type p = 0; p = s.find(".A\n.A\n\n", p); if (p == string::npos) { return 1; } } </pre>

	<pre> return 0; } </pre>
11. Verify proper tracking of Player movement after standing	<pre> UNIX COMMAND LINE: g32 main.cpp Arena.cpp Game.cpp Previous.cpp Player.cpp Robot.cpp utilities.cpp main.cpp: #include <string> #include "Arena.h" #include "Player.h" #include "Previous.h" #include "globals.h" #include <sstream> #include <iostream> using namespace std; class StreambufSetter { public: StreambufSetter(ios& str, streambuf* sb) : stream(str), oldsb(str.rdbuf(sb)) {} ~StreambufSetter() { stream.rdbuf(oldsb); } private: ios& stream; streambuf* oldsb; }; int main() { ostringstream oss; StreambufSetter ssout(cout, oss.rdbuf()); Arena a(2, 2); a.addPlayer(1, 1); a.player()->stand(); a.thePrevious().showPreviousMoves(); string s = oss.str(); string::size_type p = 0; p = s.find("A.\n..\n\n", p); if (p == string::npos) { return 1; } </pre>

	<pre> } return 0; } </pre>
12. Verify proper tracking of Player movement after moving and standing	<p>UNIX COMMAND LINE:</p> <pre> g32 main.cpp Arena.cpp Game.cpp Previous.cpp Player.cpp Robot.cpp utilities.cpp </pre> <p>main.cpp:</p> <pre> #include <string> #include "Arena.h" #include "Player.h" #include "Previous.h" #include "globals.h" #include <sstream> #include <iostream> using namespace std; class StreambufSetter { public: StreambufSetter(ios& str, streambuf* sb) : stream(str), oldsb(str.rdbuf(sb)) {} ~StreambufSetter() { stream.rdbuf(oldsb); } private: ios& stream; streambuf* oldsb; }; int main() { ostringstream oss; StreambufSetter ssout(cout, oss.rdbuf()); Arena a(2, 2); a.addPlayer(1, 1); a.player()->move(RIGHT); a.player()->stand(); a.player()->move(DOWN); a.thePrevious().showPreviousMoves(); string s = oss.str(); string::size_type p = 0; p = s.find(".B\n.A\n\n", p); if (p == string::npos) { </pre>

	<pre> return 1; } return 0; } </pre>
13. Verify proper tracking of Player movement after moving, standing, and shooting	<p>UNIX COMMAND LINE: g32 main.cpp Arena.cpp Game.cpp Previous.cpp Player.cpp Robot.cpp utilities.cpp</p> <p>main.cpp:</p> <pre> #include <string> #include "Arena.h" #include "Player.h" #include "Previous.h" #include "globals.h" #include <sstream> #include <iostream> using namespace std; class StreambufSetter { public: StreambufSetter(ios& str, streambuf* sb) : stream(str), oldsb(str.rdbuf(sb)) {} ~StreambufSetter() { stream.rdbuf(oldsb); } private: ios& stream; streambuf* oldsb; }; int main() { ostringstream oss; StreambufSetter ssout(cout, oss.rdbuf()); Arena a(2, 2); a.addPlayer(1, 1); a.player()->move(RIGHT); a.player()->stand(); a.player()->shoot(LEFT); a.player()->move(DOWN); a.thePrevious().showPreviousMoves(); string s = oss.str(); string::size_type p = 0; p = s.find(".C\n.A\n\n", p); } </pre>

	<pre> if (p == string::npos) { return 1; } return 0; } </pre>
14. Verify code does not compile because Robot.h missing	<p>UNIX COMMAND LINE: g32 main.cpp Arena.cpp Player.cpp Robot.cpp</p> <p>main.cpp:</p> <pre> #include "Player.h" #include "Arena.h" int main() { Arena a(10, 20); Player p(&a, 2, 3); Robot r(&a, 1, 1); } </pre>
15. Verify code does not compile because Arena.h missing	<p>UNIX COMMAND LINE: g32 main.cpp Player.cpp Arena.cpp Robot.cpp</p> <p>main.cpp:</p> <pre> #include "globals.h" #include "Robot.h" #include "Player.h" int main() { Arena a(10, 10); } </pre>
16. Verify code does not compile because Previous does not have default constructor defined	<p>UNIX COMMAND LINE: g32 main.cpp Previous.cpp</p> <p>main.cpp:</p> <pre> #include "Previous.h" int main() { Previous p; } </pre>