

Thy Dungeonman⁺⁺: A Dynamically-allocated, Polymorphic Text Adventure in C++

Andrew Rogers

Crazy Go Nuts University

Author Note

Andrew Rogers, Computer Science Department, Crazy Go Nuts University.

Contact: tdlovesyou@cgnuonline-eniversity.edu, @tuxlovesyou (Twitter)

Game Code: <https://github.com/tuxlovesyou/tdm-plusplus>

Abstract

Thy Dungeonman⁺⁺ is a port of the text adventure game *Thy Dungeonman* (Chapman & Chapman, 2004; poorlydrawnplato et al., 2019) made by the fictional electronic entertainment company Videlectrix (InterruptorJones, 2020; Videlectrix, 2009). This port from the game’s original Adobe/ShockWave Flash incarnation to C++ has been written to fully take advantage of modern C++ language and Standard Template Library (STL) features like lambdas, `unordered_maps`, dynamic memory management, and polymorphism throughout its code base.

Using these features greatly improved the efficiency of game’s entities, locations, and text parsing, while also decreasing the code’s complexity and verbosity. The game’s main text parser was implemented solely using simple conditional branches fed by the hashing facilities inherent in the `std::unordered_map` datatype, to form a very simple parsing tree.

Keywords: C++, text parsing, video games, dynamic memory, polymorphism

Thy Dungeonman⁺⁺: A Dynamically-allocated, Polymorphic Text Adventure in C++

Objective

The aim of the project detailed in this paper is to create a faithful C++ port of the Flash-based *Thy Dungeonman* originally made by Videlectrix (*Figure 2*), tentatively named **Thy Dungeonman⁺⁺** (at least until a C&D letter shows up in the mail from *The Brothers Chaps*) using dynamically-allocated, polymorphic data structures and objects. By the end of the project, a playable game binary **tdm** should be produced that can parse text in a similar fashion to the original. Additionally, the resultant game's code should be portable between all platforms where a standards-compliant C++14 compiler and accompanying Standard Template Library (STL) are present. For the sake of brevity, the new game will henceforth be referred by its compiled binary name, **tdm**, in this paper (for the most part).



Figure 1. A use case diagram. Video game (left) administering fun to a user (right)



Figure 2. The Videlectrix developers hard at work.

Development Methodology

As mentioned earlier, `tdm` has been implemented with dynamically-allocated, polymorphic objects, but what does that actually mean? Consider the two parts of the following statement. Dynamic memory allocation in conventional C++ parlance means that rather than building software using variables that hold or point to data of a predetermined, fixed-size, memory can be allocated at runtime (Cplusplus.com, 2020a). This allows one to allocate memory of a size only known at runtime, like in the following example that allocates a variable-length `int` array of a user-specified size¹:

```
[cling]$ #include <iostream>
[cling]$ int *some_numbers;
[cling]$ int size = 0;
[cling]$ std::cout << "How many numbers? "; std::cin >> how_many;
How many numbers? 12
[cling]$ some_numbers = new int[how_many];
[cling]$ std::cout << "Enter " << how_many << " number" \
[cling]$          << ((how_many > 1) ? "s:" : ":") << '\n';
Enter 12 numbers:
[cling]$ for (int i = 0; i < size; i++) {
[cling]$ ?   std::cout << '#' << (i + 1) << ": ";
[cling]$ ?   std::cin >> some_numbers[i];
[cling]$ ?   }
#1: 22
#2: -27
#3: 33
```

¹ The following example is run in CERN's excellent² Cling LLVM-powered C++ REPL/shell (CERN, 2018; De Simone, 2015)

² Citation needed.

```
#4: 12
#5: 33
#6: 23
#7: 23
#8: 23
#9: 23
#10: 23
#11: 23
#12: 23
[cling]$
```

You can free the memory allocated in the above example by using the `delete[]` operator (Cplusplus.com, 2020a). You can see an obvious use of this style of memory allocation strategy in `tdm`'s `src/main.cpp`'s `main()`'s first glance, albeit in a somewhat simpler way:

```
int main(void) {
    Game *game;
    bool play_again = true;

    while (play_again) {
        game = new Game();

        play_again = game->play();

        // DELETED!!!
        delete game;
    }
}
```

```
    }  
  
    return 0;  
}
```

In addition to being used in allocating memory of arbitrary sizes determined at runtime, the dynamic memory facilities in C++ can also be utilized to enable an oft-used superpower: the ability to allocate object memory belonging to a derived class on a pointer of its abstract base class's type. This allows one to store any number of **classes** and/or **structs** in a single data structure, so long as they are derived from the same base class. This is what is referred to in C++ (and in many other object-oriented programming(Frankenmint, 2016) languages) as **polymorphism** (Cplusplus.com, 2020b): the end result of combining the power of dynamic memory allocation with the power of inheritance inheritance(GeeksforGeeks, n.d.). Polymorphism can be leveraged to write code that is powerful and concise without being overly repetitious.

In **tdm**, the above C++ development strategies ended up being used quite heavily. A casual read through the source code will reveal a quite a few uses of polymorphism, from the **Game** class, all the way to the innermost derived **Item** classes that live within the various intermediary **Room** classes by reference as the **Room*** values within a **std::unordered_map<string, Room*>**, named **items**. The various **Room** classes themselves are also derived from a base **Room** class are stored by reference within a **std::vector<Room*>** container named **rooms**.

Another item of note: to easily access of the state of the game and call essential mutator methods on the core **Game** from actions triggered by **Items** within **Rooms** in a way that didn't completely invalidate the entire application's central design philosophy, a hack had to be devised, one that allowed the innermost **Rooms** and **Items** to be able to share

common utility functions with the `Game` class without awkwardly moving functionality to the `Item` class, adding unnecessary extra complexity to the game’s design in the process.

Below is this very hack, dubbed the `GameWumpus` class:

```
class GameWumpus {  
    public:  
        void gg(Game *);  
  
    protected:  
        Game *game;  
};
```

All `Rooms` and `Items` have the DNA of a `GameWumpus` inside of them, thanks to the magic of inheritance. After a `Room` or an `Item` has been created, its `void gg(Game *)` method (short for “give game”) must be called before it can be used, to ensure full functionality. This hack allows public attributes and commonly-used utility methods belonging to the main `Game` object to be shared and used by all of the various actions triggered throughout the codebase that need them. Some inner `Room` and `Item` methods need to have additional access to `private Game` attributes, hence they have been declared as `friends`³ of the main `Game` class, making the `Game` class by far and away the most popular of all of `tdm`’s classes [sic].

One of the more notable shared utility functions unlocked through the power of the `GameWumpus` is `Game`’s `sayTxt` method. This method is responsible for dynamically interpolating the special embedded format specifiers embedded in many of the in-game messages with useful data from the running `Game`:

³ friend declarations are added to C++ classes when their pals (other friendly methods) need access to attributes and methods that are usually private. (Cplusplus.com, 2019a)

```
void Game::sayTxt(const string *_txt, vector<string> &args) const {
    for (auto c = _txt->begin(); c != _txt->end(); ++c) {
        if (*c == '%') {
            switch (*++c) {
                case '%':
                    cout << '%';
                    break;
                case 's':
                    cout << score;
                    break;
                case 'A':
                    sayAnA(args);
                    break;
                case 'a':
                    sayArgs(args, 1);
                    break;
                case 'v': {
                    string verb = args.at(0);
                    if (verb == "gimme")
                        verb = "get";

                    cout << verb;
                    break;
                }
            }
        } else {
            putchar(*c);
        }
    }
}
```



```
    }  
}  
    putchar( '\n' );  
}
```

Another honorable mention is the utility function called by the aforementioned `Game::sayTxt()` is `Game::sayAnA()`, a method that handles the `%A` format specifier within game text, used for checking if the non-existent entity that the player has foolishly tried to TALK to should be mentioned in the resultant insulting error message with an “a” or an “an” article:

```
inline void Game::sayAnA(vector<string> &args) const {  
    char cons_vowel = args.at(1).at(0);  
    switch (cons_vowel) {  
        case 'a': case 'A':  
        case 'e': case 'E':  
        case 'i': case 'I':  
        case 'o': case 'O':  
        case 'u': case 'U':  
            cout << "an";  
            return;  
        }  
        cout << 'a';  
    }  
}
```

Parsing of User Input

Thanks to abilities granted by `std::basic_istream`, the output of the interactive game prompt's `std::getline` can iterate over space-separated arguments, to yield a `std::vector<string>` of the arguments, all without the need of writing code for scanning the input character-by-character or writing code that trims any leading or trailing whitespace from the aforementioned input (à la Perl's `chomp()`⁴):

```
inline vector<string> Game::getArgs(const string prompt) {
    vector<string> args;
    string line = "";

    while (line.length() == 0) {
        cout << prompt;
        getline(cin, line);

        if (cin.eof()) {
            cout << endl;
            exit(0);
        } else if (cin.bad()) {
            cout << endl;
            exit(1);
        }
    }

    istringstream astream(line);
    while (true) {
```

⁴ PerlDOC, 2020

```
    string arg;

    if (astream >> arg)
        args.push_back(arg);

    else
        break;

}

return args;

}
```

Once the arguments are separated, they get handed to the great and terrible `Room::parseCmd(vector<string> &args)`. Before it does any parsing, though, it converts the list of arguments to lowercase using the lambda-powered⁵ `void Game::lc(string &io)` from within a range-based `for` loop (Cppreference.com, 2019b):

```
void Game::lc(string &io) {
    transform(io.begin(), io.end(), io.begin(),
        [](unsigned char c){ return tolower(c); });
}
```

```
void Room::parseCmd(vector<string> &args) {
    if (args.size() < 1)
        return;
}
```

⁵ Lambdas are small anonymous functions or subroutines that can be inserted inline in scope, often producing useful output that can be captured. (Cppreference.com, 2020)

```
vector<string> myargs(args);  
  
for (string &arg : myargs) { game->lc(arg); }  
  
...
```

Most of `tdm`'s parsing of textual user input (taken exclusively from `STDIN`) is handled by the monolith known as `Room::parseCmd`. Through the magic of `std::unordered_maps` in the C++ STL⁶, and the relative simplicity of the source material (at least in *this* version of *Thy Dungeonman* from 2004), the entire parser has been able to be implemented with nary a regular expression in sight (as fun as those may be). To parse commands, `Room::parseCmd` runs the lowercase version of the first argument (the verb/action to perform) from the list of arguments through an `unordered_map` that yields an integer value pertaining to an enumerated parser opcode keys. The integer value from the `unordered_map` is fed over into a switch that hands control over to the remainder of the parsing logic that lives at the end of its conditional jump. All branches of the conditional jump have both the normalized lowercase and mixed-case versions of the argument list for in scope for parsing and display operations, respectively. Routing commands in this fashion affords the ability to map multiple verbs to the same action while also giving subsequent parsing operations ample flexibility. Putting function pointers in the `unordered_map` was also considered, but it was scrapped due to the inflexibility of being forced to implement all subsequent parsing suboperations through functions of the same type.

Here are some snippets of the first leg of the text parser:

```
enum cmd_ops { DESC_OP = 1, DIE_OP, DANCE_OP,  
               GET_OP, GO_OP, LOOK_OP, TALK_OP,  
               GIVE_OP, SMELL_OP, SCORE_OP, PWD_OP };
```

⁶ Standard Template Library

```
unordered_map<string, int> cmd_map = {  
    {"help", DESC_OP},  
    {"helpeth", DESC_OP},  
    {"look", LOOK_OP},  
    {"die", DIE_OP},  
    {"dance", DANCE_OP},  
    {"get", GET_OP},  
    // lots of GET_OP aliases...  
    {"talk", TALK_OP},  
    {"give", GIVE_OP},  
    {"smell", SMELL_OP},  
    {"sniff", SMELL_OP},  
    {"go", GO_OP},  
    {"score", SCORE_OP},  
    {"pwd", PWD_OP},  
};
```

Some locations, like DENNIS with its navigation commands, are special cases that must be accounted for, in order to remain faithful to the original game:

```
if (items["jamberjam"])  
    cmd_map["not"] = GO_OP;
```

Down yonder is the main command-routing **switch** statement (where the all of the magic happens):

```
{ // (Extra {} for more explicit scoping and memory alloc.)
  auto cmd_id = cmd_map[myargs.at(0)];
  if (cmd_id) { // If the command entered is hashed...
    switch (cmd_id) {
      case LOOK_OP:
        if (args.size() == 1) {
          desc(); // Repeat the current Room's description.
        } else {
          Item *looker = items[myargs.at(1)];
          if (looker) {
            looker->itm_look();
          } else {
            game->sayCmd(LOOK);
          }
        }
        break;
      ...
    }
  } else { // If the command entered isn't hashed...
    game->sayCmd(UNKNOWN);
  }
}
```

Results

Before the game can run, it must first be built with GNU Autotools using the project's included configuration:

```
~/src/cs200-poly$ autoreconf --install
~/src/cs200-poly$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
...
~/src/cs200-poly$ make
make all-recursive
make[1]: Entering directory '/home/anon/src/cs200-poly'
Making all in src
make[2]: Entering directory '/home/anon/src/cs200-poly/src'
g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT Game.o -MD -MP -MF
.deps/Game.Tpo -c -o Game.o Game.cpp
mv -f .deps/Game.Tpo .deps/Game.Po
g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT main.o -MD -MP -MF
.deps/main.Tpo -c -o main.o main.cpp
mv -f .deps/main.Tpo .deps/main.Po
g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT Item.o -MD -MP -MF
.deps/Item.Tpo -c -o Item.o Item.cpp
mv -f .deps/Item.Tpo .deps/Item.Po
g++ -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT Rooms.o -MD -MP -MF
.deps/Rooms.Tpo -c -o Rooms.o Rooms.cpp
mv -f .deps/Rooms.Tpo .deps/Rooms.Po
g++ -g -O2 -o tdm Game.o main.o Item.o Rooms.o
make[2]: Leaving directory '/home/anon/src/cs200-poly/src'
make[2]: Entering directory '/home/anon/src/cs200-poly'
make[2]: Leaving directory '/home/anon/src/cs200-poly'
make[1]: Leaving directory '/home/anon/src/cs200-poly'
```

Running the compiled game:

```
"# " # # # # "#= # # #"#"#"#"#"#"="# #"#"#"# . .
# #"# # # # # # # #"#"# # # # # #"# # # =#=#=#
" " " " " " " " " " " " " " " " " " " " " " " "
```

```

. . . . . m m mmm m m . . . . .
. . . . . "m m" #" "# # # . . . . .
. . . . . #m# # # # # . . . . .
. . . . . "# "#m#" "mm"# . . . . .
. . . . . m" . . . . .
. . . . . "" . . . . .

. . . . . mmm m mm mmm . . . . .
. . . . . " # #" "# # . . . . .
. . . . . m"" "# # "" . . . . .
. . . . . "mm"# # "#mm" . . . . .

. . . . . "#" . . . . .
. . . . . #.#.# . . . . .
. . . . . "#"#.#.# . . . . .
. . . . . " ".# . . . . .
. . . . . " . . . . .

```


What wouldst thou deau?
>

From this point, the rest of the game ostensibly works, though an entire walkthrough will not be delivered in this section, so as not to spoil the entire game for the reader. Instead, a few noteworthy features of the game will be highlighted in this section (*continued on the next page*).

New GET verb aliases for the player to use and discover:

```
Ye find yeself in yon dungeon.  Ye see a SCROLL.  
Behind ye scroll is a FLASK.  Obvious exits are  
NORTH, SOUTH, and DENNIS.
```

```
What wouldst thou deau?
```

```
> snatch yon flask
```

```
Ye cannot get ye FLASK. It is firmly bolted to a  
wall which is bolted to the rest of the dungeon  
which is probably bolted to a castle. Never you mind.
```

```
What wouldst thou deau?
```

```
> yeet yon flask
```

```
Ye cannot get ye FLASK. It is firmly bolted to a  
wall which is bolted to the rest of the dungeon  
which is probably bolted to a castle. Never you mind.
```

Appropriate verb propagation to error messages that are triggered by new GET_OP aliases:

```
What wouldst thou deau?
```

```
> take red steckled elbermung
```

```
Thou cannotst take that. Quit making stuffeth up!
```

```
What wouldst thou deau?
```

```
> grab Da Huuuuuudge
```

```
Thou cannotst grab that. Quit making stuffeth up!
```

```
What wouldst thou deau?
```

```
> gimme deep sea fangly fish
```

```
Thou cannotst get that. Quit making stuffeth up!
```

Correct a/an article selection when the player attempts to GIVE something that they don't have:

```
What wouldst thou deau?  
> give Ab-Abber 2000  
Thou don'tst have an Ab-Abber 2000 to give. Go back to your  
tiny life.  
  
What wouldst thou deau?  
> give Lappy 486  
Thou don'tst have a Lappy 486 to give. Go back to your  
tiny life.
```

Just like in the original Flash game, the **SOUTH** room is stateful in its insults:

```
What wouldst thou deau?  
> go south  
You head south to an enbankment. Or maybe a chasm. You  
can't decide which. Anyway, ye spies a TRINKET.  
Obvious exits are NORTH.  
  
What wouldst thou deau?  
> look  
Ye stand yeself close to a yet-unnamed escarpment.  
Nonetheless, ye spies a TRINKET. Obvious exits are  
NORTH.  
  
What wouldst thou deau?  
> get trinket  
Ye getsts yon TRINKET and discover it to be a bauble.  
You rejoice at your good fortune. You shove the  
TRINKET in your pouchel. It kinda hurts.  
  
What wouldst thou deau?  
> go north  
Ye find yeself in yon dungeon. Ye see a SCROLL.  
Behind ye scroll is a FLASK. Obvious exits are  
NORTH, SOUTH, and DENNIS.  
  
What wouldst thou deau?  
> go south
```

```
Ye stand high above a canyon-like depression. Obvious
exits are NORTH.
```

```
What wouldst thou deau?
```

```
> get trinket
```

```
Sigh. The trinket is in thou pouchel. Recallest thou?
```

```
What wouldst thou deau?
```

```
> help
```

```
Thou hangeth out at an overlook. Obvious exits are
NORTH. I shouldn't have to tell ye there is no TRINKET.
```

```
What wouldst thou deau?
```

```
>
```

Live game object deletion appear and new game creation appear to all work cleanly, though the code has yet to be extensively tested with `valgrind` or `gdb` in conjunction with a Bash/Python fuzzer, so it cannot be said with 100% confidence that it *does not* leak *any* memory... With that being said, here is an example of **Thy Dungeonman**⁺⁺'s live reload/respawn in action:

```
What wouldst thou deau?
```

```
> die
```

```
That wasn't very smart. Your score was: -95.
```

```
Play again? [Y/N] y
```

```
"#" # # # #   #"= # # #"# #" " #" " #"# #"# #=# #"# #"#   .   .
#   #"# #   # # # # # # # # #" " # # # # # # #"# # # #=# #=#
"   "   "   "   ""   "" "" " " "" "" "" "" "" "" "" " " " "   "   "
```

```

      /\      /\      /\
      ||/----\||      ||
      \_-----_/      ||
      / o o \         ||
      /  ||  \        o__||__o
      / ---- \        \____/

```

```
~=Press the ENTER key to enter yon dungeon=~
```

```

.....
.....
.....
.....      m      m      mmm      m      m      .....
.....      "m m"    #"  "#    #      #      .....
.....      #m#      #      #    #      #      .....
.....      "#       "#m#"    "mm"#      .....
.....      m"                           .....
.....      ""                           .....

```

...	mmm	m	mm	mmm	...
...	" #	#	"	#	" #
...	m" " " #	#		#	" " " "
...	"mm" #	#		"	#mm"

```

....."#".....
.....#.#.....
....."##.#.....
....." "#.....
....."

```

[illegible]

Ye find yeself in yon dungeon. Ye see a SCROLL. Behind ye scroll is a FLASK. Obvious exits are NORTH, SOUTH, and DENNIS.

What wouldst thou deau?

>

Analysis

After much trial and tribulation, it can still be said that you still cannot GET ye FLASK, and after all that has been learned here, it should be readily apparent that this is as it should be, nay, as it always was *meant* to be. All joking aside, it can definitely be said that the goals set at the onset of this project's conception were met (and perhaps even exceeded): a native C++ port of the original Flash/ActionScript *Thy Dungeonman* has been made, written to take full advantage of the dynamic and polymorphic features of C++, where it has been deemed logical to do so. Thanks to the flexibility of the C++ STL's data structures and built-in algorithms, implementing this entire text adventure game from scratch is just a matter of forming a cohesive mental model of the game's modes of operation and making way for sensible base classes with which to build the rest of the game upon. For the most part, the polymorphism and data types given by C++ allowed the rest to fall in place, once enough of said basic framework was put in place, and the vision of the game's implementation only became clearer as the work continued to progress.

In closing, it can be said that this project was a resounding success: the parser only be said to be an improvement on the original ActionScript one, while still retaining all of the original game's charm, even managing to take advantage of C++'s polymorphic, dynamic feature set along the way. The seed of a great *Thy Dungeonman* port has been planted, one that can builds on Linux, macOS, and even Windows (using MSYS2, MinGW, or Cygwin, recommended in that order). With the base game and parser done, the game can go in any number of directions, so long as I can get the blessing of *Videlectrix*, aka. *The Brothers Chaps*. Perhaps one day there will be a version of this port using *ncurses*, *SDL*, or even something more exotic involving physical hardware!

References

- CERN. (2018). Cling. Retrieved May 2, 2020, from <https://root.cern.ch/cling>
- Chapman, M. [Matt], & Chapman, M. [Mike]. (2004). Thy dungeonman! thy hunger!
Retrieved May 1, 2020, from <http://homestarrunner.com/dungeonman.html>
- Cplusplus.com. (2020a). Dynamic memory. Retrieved May 2, 2020, from
<https://www.cplusplus.com/doc/tutorial/dynamic/>
- Cplusplus.com. (2020b). Polymorphism. Retrieved May 2, 2020, from
<https://www.cplusplus.com/doc/tutorial/polymorphism/>
- Cppreference.com. (2019a). Friend declaration. Retrieved May 2, 2020, from
<https://en.cppreference.com/w/cpp/language/friend>
- Cppreference.com. (2019b). Range-based for loop (since c++11). Retrieved January 31,
2020, from <https://en.cppreference.com/w/cpp/language/range-for>
- Cppreference.com. (2020). Lambda expressions. Retrieved May 2, 2020, from
<https://en.cppreference.com/w/cpp/language/lambda>
- De Simone, S. (2015). Cling aims to provide a high-performance c++ repl. Retrieved May
2, 2020, from <https://www.infoq.com/news/2015/05/cling-cpp-interpreter/>
- Frankenmint. (2016). Comparing programming paradigms: Procedural programming vs
object-oriented programming. Retrieved January 31, 2020, from
<https://www.codementor.io/learn-programming/comparing-programming-paradigms-procedural-programming-vs-object-oriented-programming>
- GeeksforGeeks. (n.d.). Inheritance in c++. Retrieved May 2, 2020, from
<https://www.geeksforgeeks.org/inheritance-in-c/>
- InterruptorJones, e. a. (2020). Videlectrix. Retrieved May 1, 2020, from
<http://www.hrwiki.org/wiki/Videlectrix>
- PerlDOC. (2020). Chomp. Retrieved May 2, 2020, from
<https://perldoc.perl.org/functions/chomp.html>

poorlydrawnplato et al., e. a. (2019). Thy dungeonman. Retrieved May 1, 2020, from
http://www.hrwiki.org/wiki/Thy_Dungeonman

Videlectrix. (2009). Videlectrix home - home of our videlectrix!! Retrieved May 1, 2020,
from <http://www.videlectrix.com>