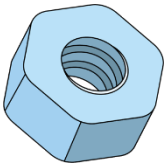
MONEY SYSTEMS IN TEXT ADVENTURES AND THEIR DESIGN CHALLENGES

Lea Stella Santner, Wilfried Elmenreich

Text adventure games, also known as interactive fiction, allow players to engage with the game world by typing commands. These games often feature intricate puzzles that necessitate correct actions and the combination of specific objects to progress. Although the concept of money is infrequently encountered within these games, there are instances where a single coin becomes instrumental in operating machinery or serves as part of a puzzle. Incorporating a monetary system into text adventures presents a formidable challenge to the genre. The introduction of money can disrupt puzzle complexity, enabling players to bypass cumbersome acquisition tasks by simply purchasing necessary objects. Despite these hurdles, some text adventures have implemented functional money systems. This paper addresses the intricate design and implementation challenges associated with introducing money systems into text adventure games. This leads to discussions about its impact on narrative choices, such as deciding between spending or retaining money, and how these decisions influence the overall storyline, as well as to technical questions regarding the implementation of the concept using a typical programming language for interactive fiction, such as Inform. As a case study, the implementation process of a money system using the Inform 6 programming language and the PunyInfom Library is presented.

Keywords: Interactive Fiction, Text Adventures, Game Engineering



1. Introduction

A text-based adventure game is a genre of computer games within the wider area of interactive fiction. While interactive fiction also includes systems where the user can make inputs via selections without entering text, the interface to the player in a text-adventure takes place by entering a text string that contains the players' intended action to the game. Will Crowther's game "Adventure" (Crowther, 1976) is considered the first text adventure and was released in 1976 for the PDP-10 mainframe computer. The game, later renamed Colossal Cave Adventure, was massively popular among the computer community in the later 1970s and kicked off an era of further text adventure games in the coming years.

As a genre, text adventures are incredibly flexible. While there is a more or less agreed set on standard verbs (cf. Plotkin, 2010), adding any other custom verbs with different effects to a game is possible. A text adventure can be implemented with standard programming languages such as C, C++ (Sutherland, 2014), Python (Johnson, 2018), or the BASIC programming language (Lampton, 1986; Montford, 2005), which used to be typically integrated into the ROM of home computers in the 1980s. A recent example of a simple text adventure written in BASIC is "Tiny Quest" (Derocher, 2024), which fits into the 3.5 kB BASIC memory of the VIC 20 (and therefore easily fits on all other systems that typically have larger memory). On the other hand, there exist dedicated programming languages and systems for text adventures, most notably Inform (Nelson, 1993), The Quill (Yeandle, 1983), and the D42 Adventure System (Lesch & Erbsland, 2014) that offer streamlined and specialized tools tailored to the unique demands of interactive storytelling.

Implementing money systems in text adventures is a challenge for several reasons: First, on a fundamental level, money comes in units, so unlike many unique objects in adventure games, one piece of money is like the other, and you can obtain multiple of them. Second, money is a means to ease trading and exchange of goods by being a joker item that can be bought or sold for any other good in a market. In the context of a text adventure, this challenges the creation of puzzles, which are an intrinsic element of these games. Money obtained in the game could be used to shortcut the need to obtain an item needed in a puzzle, thus breaking puzzle complexity. The versatility of money interactions can also affect playability: the player might spend the money on the wrong items just because it is possible, ending up with an unsolvable game state. While the last two issues require to be solved within the context of the particular setup of a text adventure, we will focus on generically implementing a money system in

Inform6. This system was chosen because it is the programming language that allows the most complexity for implementing a text adventure among the possibilities listed above. Inform 7, a newer version of the Inform language, is quite different from Inform 6. It adopts a natural-language-oriented approach to source code, which sets it apart from established systems. Because Inform 7 has a significantly different syntax and is less widely used than Inform 6, it is outside the scope of this paper.

This paper investigates the topic of money systems in text adventures. In particular, it contributes three parts: The following section presents a discussion on the challenges of integrating money systems as a game mechanic in text adventures. Subsequently, we present an overview of notable text adventure games with money systems. The third section of the paper elaborates on how money systems can be implemented in a text adventure using the programming language Inform 6. The concluding section summarizes the findings and gives an outlook on possible future work.

2. Money as a Game Mechanic

Literature on money as a game mechanic usually understands the topic as players spending real money in (mostly online) games (Wohn, 2014), or interpreting the game as a marketing instrument to advertise virtual goods (Hamari & Lehdonvirta, 2010). Besides this direct link to real money, findings suggest that even simulated gambling games using virtual currency may promote gambling with real money (Armstrong et al., 2018). In addition, in-game currencies that are connected to the real-world economy also pose a threat of being misused for money laundry and concealment of money flows (Cloward & Abarbanel, 2020).

Kinnunen, Alha, and Paavilainen (2016) further suggest that F2P gamers spending real money on their game will need to develop methods similar to those used by gamblers who frame or separate play money from other forms of money, such as money for groceries. Leijnen, Brinkkemper, and Bouwer (2015) discuss games with a money system and procedurally generated goods as a learning game for a general audience, teaching them how banks function within a market-guided economy.

A discussion on currency in fictional works[1] describes challenges regarding naming, appearance, and possibilities of obtaining money in fiction, which also apply to interactive fiction/text adventures. When naming currencies, authors must consider cultural associations to maintain coherence within their worlds and to avoid associations with existing currencies (Athans & Salvatore, 2010). Narrative challenges arise in ensuring the credibility and stability of these money systems, particularly in the face of futuristic technology or magical elements (Gliddon, 2005). Some currencies are inherently valuable within their fictional universes, serving as more than mere mediums of exchange, as seen in works like Frank Herbert's Dune series or Iain M. Banks's Consider Phlebas.

Text adventures, being a form of interactive fiction, face unique challenges related to fictional currencies. Critical questions include: Is the currency name appropriate for the world setting? Does it make sense within the context of advanced technology or magical abilities? As seen in some fictional works where currencies transcend their role as mere exchange mediums, money systems in text adventures are expected to have a similar impact, particularly given the complex nature of their puzzles. It is impractical for players to gather enough in-game money to buy all quest items outright. Instead, some items must be acquired through traditional means, with the money system playing only a partial role in item acquisition.

3. Text Adventures with Money Systems

Colossal Cave Adventure, being the first game in the genre, did not have a dedicated money system, but valuables and treasures that could be discovered in the caves, which could be seen as the predecessor of an in-game currency.

In-game currencies have been used in several text adventures, most notably in games by Infocom. Table 1 gives an overview of some examples: The names of the currencies are typically very diverse: there are Zorkmids in the classic text-based adventure game series Zork (Infocom, 1980,1981,1983) (referred to as “coin” in Zork I), Buckazoids in the interactive fiction games Planetfall (Infocom, 1983) and its sequels, the humorous use of "Flotsam and jetsam" in the text adventure The Hichhiker's Guide to the Galaxy (Infocom, 1984). Players typically

[1] https://list.fandom.com/wiki/List_of_fictional_currencies

obtain money by exploring the game world, discovering treasures, solving puzzles, and completing tasks. The money is then used to purchase goods and services (sometimes via a bribe) and advance the player's character. However, the use of money in these adventures is always limited to specific parts of a puzzle, unlike in, for example, economic simulation games where money is the principal means to purchase all relevant goods and services.

Fallen London (Failbetter Games, 2009) is a bit of a different case: it is a free-to-play text-based open-world RPG played in the web browser. In Fallen London, Echos, Pennies, and Shillings serve as the primary forms of currency in the dark and mysterious Victorian setting. Players can earn these currencies by completing storylines, engaging in trade, and participating in challenges. They are essential for purchasing goods and services and advancing the player's character. Since the game is a multiplayer game, the money also serves as a unit of exchange for trading between players.

Table 1. Text Adventures with Money Systems

Game	In-game currency
Colossal Cave Adventure (1976)	Valuables and treasures
Fallen London (2009)	Echoes, pennies, shillings
Planetfall (1983)	Buckazoids
The Hitchhiker's Guide to the Galaxy (1984)	Flotsam, jetsam
Zork (1980-1982)	Zorkmids

4. Case study: Implementation of a Money System in Inform 6

Inform 6 is a sophisticated system tailored to the creation of narrative-driven text adventure games. It offers a comprehensive suite of tools to facilitate the translation of textual descriptions into virtual worlds. At its core, Inform consists of a library of pre-defined elements and a compiler that allows authors to

construct complex textual game environments with relative ease. The foundation of Inform is its library, which includes a parser and a world model. The parser acts as the interface between the player and the game world, interpreting the commands typed and executing the corresponding actions within the virtual environment. At the same time, the world model defines a set of standard rules that govern interactions within the game, such as visibility constraints in the absence of light sources.

Since its establishment in 1993, Inform has become a significant tool for creating interactive fiction in various natural languages. It has developed into a complete software suite, including a compiler and a library, that is essential for designing games of any size. Its versatility extends beyond entertainment, finding applications in commercial game prototyping and academic contexts alike. According to the original Inform 6 page[2], Inform is used in a range of educational settings, from computer science curricula to theoretical architecture seminars. In her blog, Emily Short (2019) lists resources and examples for pedagogical uses of interactive fiction (IF) in an educational setting, covering its application in teaching English, language arts, literature, history, and foreign languages. She also provides specific examples of IF games written in Inform 6, Inform 7, and Twine. Although newer tools and languages have emerged, Inform 6, with its text-based syntax, remains relevant for text adventures due to its robustness, simplicity, and community support.

In the following, we sketch an implementation of a money system for the programming language Inform 6. Inform, being a domain-specific language for text adventures, already supports the implementation of objects that can be discovered and acquired when playing a game. A money system is more intricate because it requires objects (coins) that can be obtained multiple times and that need to be used together in case an object is purchased that costs a multitude of coins (which is usually the case).

While there exist several tutorials for implementing adventure games with Inform, this guide pertaining to money systems that are presented here has been elaborated from scratch - to our knowledge no such resource exists up to now. As a prerequisite, we assume that the reader is already familiar with the basic usage of Inform 6 and the PunyInform library. A recommended tutorial is the

[2] http://www.inform-fiction.org/introduction/index.html

Inform Designer's Manual (Nelson, 2001) and the PunyInform Manual (Berntsson & Ramsberg, 2023).

```
Constant Story      "Monetary System";
Constant Headline   "^How to implement a money system.^";
Constant STATUSLINE_SCORE; Statusline score;
Constant NO_SCORE = 0;

Constant OPTIONAL_LIST_TOGETHER;
Constant OPTIONAL_LANGUAGE_NUMBER;
Constant OPTIONAL_ALLOW_WRITTEN_NUMBERS;
Constant OPTIONAL_FLEXIBLE_INVENTORY;

Constant INITIAL_LOCATION_VALUE = Library;
Include "globals.h";

! routines here

! include PunyInform library
Include "puny.h";

! classes here

! example game code here

! routine that runs at the start of the game
[Initialise;
  print "^^That's how you do it";
];
```

Listing 1. Header section of .inf source file

You start by defining constants such as Story, Headline, etc. All constants should be defined before you include "globals.h".

The "Initialise" routine is a code block that executes certain actions when the game is initialized or started. In this case, it simply prints the message "That's how you do it" to the screen.

As the next step, it is necessary to add the routines that calculate the object's depth and its final destination (Listing 2):

```
! Calculate an object's depth in the containment hierarchy
[ ObjDepth p_obj _i;
while(p_obj) { p_obj = parent(p_obj); _i++; }
  return _i;
];

! final object destination, ensuring objects in the same group are handled
properly
[ ChooseObjectsFinal p_arr p_len _i _j _o _o2 _sg _d _d2 _other_group_pre-
sent;
#Ifdef DEBUG;
    print "*** ChooseObjectsFinal, action is ",(DebugAction) action,",
object count is ", p_len, ": ^";
#Endif;
  ! Iterate over all objects in the array
  for(_i = 0: _i < p_len: _i++) {
    _o = p_arr-->_i;
    if(_o provides same_group) {
      _sg = _o.same_group;
      _d = ObjDepth(_o);
      ! Iterate over the remaining objects in the array
      for(_j = p_len - 1: _j > _i: _j--) {
        _o2 = p_arr-->_j;
        if(_o2 provides same_group) {
          if(_o2.same_group == _sg) {
            _d2 = ObjDepth(_o2);
            ! swap if the other object is shallower in hierarchy
            if( _d2 < _d) {
              _o = _o2;
              p_arr-->_j = p_arr-->_i;
              p_arr-->_i = _o;
              _d = _d2;
            }
            ChooseObjectsFinal_Discard(_j);
            p_len--;
          } else
            _other_group_present = true;
        }
      }
      ! If no other group is present, terminate early
      if(_other_group_present == false)
        return;
      _other_group_present = false;
    }
  }
];
```

Listing 2. Routines

The overall implementation of the adventure game is in a single .inf file. The routines will be added where the comment "! routines here" is placed in Listing1.

The routine “ObjDepth” calculates the depth of an object within its object hierarchy. It starts with the given object “p_obj” and iterates through its parent objects using a while loop. For each parent object it increments a counter “_i”. Finally, it returns the value of “_i”, which represents the depth of the object within its hierarchy. Essentially, it counts how many levels of parent objects the given object has until it reaches the top parent.

The routine called “ChooseObjectsFinal” sorts objects stored in an array “p_arr” by their group membership and depth. It iterates over each object and checks if it has a property called “same_group”. If it does, it compares its depth to other objects in the same group. Objects with lower depths are moved to the front of the array. If there are objects from other groups, it continues processing, otherwise it stops and returns. The routine essentially organizes objects by group and depth.

```
! Generic coin class, used by SilverCoin and GoldCoin
Class Coin
  with
    name ',//',
    parse_name [ _w _n;
      _w = NextWord();
      if(_w == self.name) {
        _w = NextWord();
        _n++;
      }
      if(_w == 'coin')
        _n++;
      else if(_w == 'coins//p') {
        parser_action = ##PluralFound;
        _n++;
      }
      return _n;
    ];

Class SilverCoin
  class Coin,
  with
    same_group 2, ! Group identifier for same types of coins
    name 'silver',
    short_name "silver coin",
    list_together [ _obj _n;
      if(inventory_stage == 1) {
        for(_obj=parser_one: _obj ~= 0: _obj = NextEntry(_obj, parser_two)) _n++;
        print (LanguageNumber) _n, " silver coins";
        if(c_style & NEWLINE_BIT)
          new_line;
        rtrue;
      }
    ];
  Class GoldCoin
    class Coin
    with
      same_group 3, ! Group identifier for same types of coins
      name 'gold',
      short_name "gold coin",
      list_together [ _obj _n;
        if(inventory_stage == 1) {
          for(_obj=parser_one: _obj ~= 0: _obj = NextEntry(_obj, parser_two)) _n++;
          print (LanguageNumber) _n, " gold coins";
          if(c_style & NEWLINE_BIT)
            new_line;
          rtrue;
        }
      ];
```

Listing 3. Classes

The classes will be added where the comment "! classes here" is placed in Listing1.

The “Coin” class defines the behavior and attributes of coins in the game. It contains an attribute called name, which defaults to “,//” (a placeholder). It also contains a method called “parse_name”, which is responsible for parsing the name of a coin during gameplay input processing. This method checks the next

word in the input and adjusts the parsing based on matches with the name attribute, “coin”, or “coins//p” (plural form). Finally, it returns the number of words used during parsing. The “Coin” class thus facilitates accurate recognition and processing of coin-related commands by handling variations in coin names.

The “SilverCoin” and the “GoldCoin” classes extend the functionality of the “Coin” class, specifically for representing silver and gold coins in the game. They introduce attributes such as “same_group”, “name” and “short_name” to define coins. They also include a method called “list_together”, which handles the inventory listing of silver and gold coins, ensuring accurate identification and labeling in the game environment.

```
Object Library "The Library"
  with
    description "You are in a library."
  has light;

Object -> Table "table"
  with
    name 'table',
  has supporter open enterable;

Object -> -> Box "box"
  with
    name 'box',
    inside_description "It feels so nice, standing in the box.",
  has container open openable enterable;


GoldCoin -> -> -> GoldCoin1;
GoldCoin -> -> -> GoldCoin2;
GoldCoin -> -> -> GoldCoin3;

SilverCoin -> -> -> SilverCoin1;
SilverCoin -> -> -> SilverCoin2;
SilverCoin -> -> -> SilverCoin3;
```

Listing 4. Example game code

Additional game code will be added where the comment "! example game code here" is placed in Listing 1.

The "Library" object represents a location within the game environment that is described as a library. It contains a description that provides information about the environment, stating "You are in a library." Additionally, it is specified to have light, indicating that the location is bright. The "Table" object represents a

table in the game environment. It is defined with the name 'table' and specified to be a supporter, indicating that it can hold other objects. It is also open and enterable, suggesting that characters or items can be placed on or inside it during gameplay. The "Box" object is described as a box in the game. It is named "box". The inside of the box is further described with the text "It feels so nice, standing in the box." This object is specified as a container, indicating that it can hold other objects. It is set to be open, openable, and enterable, suggesting that characters or items can be placed inside it during gameplay and that it can be interacted with by opening it.

The provided example of the implementation of a monetary system in Inform 6 defines classes such as “Coin”, “SilverCoin”, and “GoldCoin”, allowing for the creation of various types of currency within the game world. Each coin is associated with its respective class ("GoldCoin" or "SilverCoin") and is uniquely identified by a numerical suffix. These instances represent individual coins that can be interacted with separately within the game. The presented approach enables efficient tracking and management of similar coins within the player's inventory, with clear inventory management features. Players can seamlessly interact with monetary objects, including picking them up, dropping them, and potentially using them for in-game transactions or interactions. Descriptive content for monetary objects is supported, enhancing player immersion and understanding of the game's economic elements.

Although the code example shows proficiency in handling basic monetary systems, it needs to be extended when dealing with more complex economic interactions. Advanced economic features, such as intricate trading mechanisms or dynamic market systems, may require additional implementation beyond the capabilities provided in the code. The debugging and testing tools for the monetary system within the code snippet may be limited, which can complicate the identification and resolution of issues related to monetary interactions.

5. Conclusion

This paper explored strategies for integrating the concept of currency into text adventures without compromising puzzle-solving intricacies. In particular, a challenge arises when puzzles traditionally have a singular solution whereas a currency system introduces a multitude of decision points. Generally, money systems should be introduced with care, even if they do not involve real money. Our literature research in Section 2 suggests that even simulated gambling

systems can negatively affect gamers in real life. For game economies that are connected to real-world money, there is the risk of adverse effects on players spending significant money in order to perform well in-game. However, the described effects typically apply to online games, where players employ all kinds of means to outperform each other.

Money systems in interactive fiction games come with distinct challenges. They need to align with the theme and be carefully managed. Players should not gather enough in-game money to purchase all quest items at once. Traditional methods should still be necessary for obtaining several items, with the money system playing a minor role.

The case study in Section 4 shows how such a money system can be implemented in Inform 6. While the overall effort is concise, the implementation steps are intricate due to the complexity and versatility of the Inform programming language. The guide included in this paper thus is intended to serve as an enabler for implementing text adventures with meaningful and game-enhancing money system mechanics. In future work, we expect the implementation of money systems in Inform to become more accessible by providing the necessary steps in the form of a library or being supported by a tool. What remains further open are test strategies for games with money systems. The versatility of money introduces complexity for both players, who may welcome new possibilities, and testers, who will face considerable challenges. Therefore, a future convenient-to-use money system will benefit from an automated testing approach.

Acknowledgments

The work leading to these results has received funding from the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology Austria (BMK), from the HTBLuVA Villach, and its parent association.

About the Authors

Lea Stella Santner is a student of the HTL Informationstechnologie in Villach and an associate student of psychology and informatics at the University of Klagenfurt. Her interests cover Software and Web development, as well as game engineering, virtual worlds, character design, storytelling, and game graphics.

Lea was working with the Smart Grids group at the University of Klagenfurt in July 2023 on the analysis implementation of different text adventure systems. The work involved devising an Italian and German grammar module for the PunyInform library and the evaluation of money systems in interactive fiction games.

LinkedIn: lea-santner-45a50228b

Wilfried Elmenreich is professor of Smart Grids at the Institute of Networked and Embedded Systems at the Alpen-Adria-Universität Klagenfurt, Austria. His research interests include intelligent energy systems, self-organizing systems, and technical applications of swarm intelligence. Wilfried Elmenreich is a member of the Senate at the Alpen-Adria-Universität Klagenfurt, Counselor of the IEEE Student Branch, and is involved in the master program on Game Studies and Engineering. He is the author of several books and has published over 200 articles in the field of networked and embedded systems. Elmenreich researches intelligent energy systems, self-organizing systems, and technical applications of swarm intelligence.

LinkedIn: wilfriedelmenreich
Website: https://mobile.aau.at/~welmenre/

References

Nelson, G. (2001). The Inform Designer's Manual, Fourth Edition.

Crowther, W. (1976). *Adventure* (also known as *Colossal Cave Adventure*) PDP-10 game published via ARPANET.

Johnson P. (2018). Make Your Own Python Text Adventure. Apress.

Sutherland, B. (2014). Text Adventure. In: C++ Game Development Primer. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-0814-4_6

Lampton C. (1986). How to Create Adventure Games. Ed. Franklin Watts; Library Binding Edition

Montford N. (2005). Twisty Little Passages - An Approach to Interactive Fiction. MIT Press.

Plotkin A., Albaugh L. (2010). How To Play Interactive Fiction (An entire strategy guide on a single postcard). People's republic of interactive fiction. http://pr-if.org/doc/play-if-card/

Derocher, R.J. (2024). Tiny Quest. Computer Game. Binary Legends. https://csdb.dk/release/?id=239276

Short, E. (2019). Mailbag: Pedagogical Uses of IF in the Classroom. Blog: Emily Short's Interactive Storytelling. Retrieved from https://emshort.blog/2019/09/10/mailbag-pedagogical-uses-of-if-in-the-classroom/

Nelson G. (1993). Inform. Programming Language. https://ganelson.github.io/inform-website/

Berntsson J. & Ramsberg F. PunyInform. Library for Inform 6. https://github.com/johanberntsson/PunyInform

Yeandle G. (1983). The Quill. Computer Software. Gilsoft.

Lesch, S.& Erbsland T. (2014) D42 Adventure System: Klassische Adventures selbst entwickeln für den Commodore 64/128. BoD – Books on Demand.

Wohn, D. Y. (2014). Spending real money: Purchasing patterns of virtual goods in an online social game. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14) (pp. 3359–3368). Association for Computing Machinery. https://doi.org/10.1145/2556288.2557074

Hamari, J., & Lehdonvirta, V. (2010). Game design as marketing: How game mechanics create demand for virtual goods. International Journal of Business Science & Applied Management, 5(1), 14–29. https://ssrn.com/abstract=1443907

Armstrong, T., Rockloff, M., Browne, M., et al. (2018). An exploration of how simulated gambling games may promote gambling with money. Journal of Gambling Studies, 34, 1165–1184. https://doi.org/10.1007/s10899-018-9742-6

Cloward, J. G., & Abarbanel, B. L. (2020). In-Game Currencies, Skin Gambling, and the Persistent Threat of Money Laundering in Video Games. UNLV Gaming Law Journal, 10(1), Article 6. Retrieved from https://scholars.law.unlv.edu/glj/vol10/iss1/6

Kinnunen, J., Alha, K., & Paavilainen, J. (2016). Creating play money for free-to-play and gambling games. In Proceedings of the 20th International Academic Mindtrek Conference (AcademicMindtrek '16) (pp. 385–392). Association for Computing Machinery. https://doi.org/10.1145/2994310.2994336

Leijnen, S., Brinkkemper, P., & Bouwer, A. (2015). Generating game mechanics in a model economy: A MoneyMaker Deluxe case study. Paper presented at the 6th Workshop on Procedural Content Generation in Games (PCG 2015), Pacific Grove, United States.

Athans, P., & Salvatore, R. A. (2010). The Guide to Writing Fantasy and Science Fiction. Adams Media.

Gliddon, G. (2005). The Greenwood encyclopedia of science fiction and fantasy, Volume 2. Greenwood.

Infocom. (1980). Zork I [Computer game]. United States: Infocom (Developers: Tim Anderson, Marc Blank, Bruce Daniels, Dave Lebling).

Infocom. (1981). Zork II [Computer game]. United States: Infocom (Developers: Tim Anderson, Marc Blank, Bruce Daniels, Dave Lebling).

Infocom. (1982). Zork III [Computer game]. United States: Infocom (Developers: Tim Anderson, Marc Blank, Bruce Daniels, Dave Lebling).

Infocom. (1984). The Hitchhiker's Guide to the Galaxy [Computer game]. Cambridge, MA: Infocom (Developers: Douglas Adams, Steve Meretzky).

Infocom. (1983). Planetfall [Computer game]. Cambridge, MA: Infocom (Developer: Steve Meretzky).

Failbetter Games. (2009). Fallen London. [Online Game]. Failbetter Games.

Nelson, G. (2001). *The Inform Designer's Manual* (4th ed.). (G. Rees, Ed.). Retrieved from https://www.inform-fiction.org/manual/DM4.pdf

Berntsson, J., & Ramsberg, F. (2023). *PunyInform: An Inform library for writing compact and fast text adventures* (Version 5.3). Retrieved from https://github.com/johanberntsson/PunyInform/wiki/manual