"Dr. Rorhbaugh and Mr. Tigerram"
Code Documentation

Nathan R Chaney
Zachary J Felix
Rebekah P Smith

February 18, 2015

Contents

Part I

Static Predicates

Chapter 1

Describing Things

1.1 Generic Predicates

1.1.1 `description(Name, Type, ShortName, LongName)`

`description/4` describes the object referred to by `Name`. Current values for `Type`, with their meanings, are listed below. `ShortName` is the name that will generally be displayed for this object, and `LongName` will generally be displayed upon `look`ing at the object.

`building` – a building that the player can be standing outside of

`maze` – a room inside representing a maze

`place` – an outside place that is not a specific building

`room` – a room inside a building

`waypoint` – a location outside that is relatively unimportant other than that it is on the way to somewhere else

`door` – a door that can be locked to prevent progress

`key` – a key that can be used to unlock a door

`sa` – a stuffed animal

`dead` – a dead thing

`object` – a general object of some sort that can be picked up

`fakeobject` – a general object of some sort that cannot be picked up

`lightsource` – an object that provides light

`person` – a specific person who can be talked to and might wander around

`monster` – some sort of entity that is probably less specific than a person and might hinder the player's progress

1.1.2 `get_name(Thing, Name)`

`get_name/2` is an easier way to get the name of an object from `description/4`, and is set up to be used with `select_thing/3`.

1.1.3 helplines(List)

`List` contains the list of lines to print in the help message.

1.1.4 look_at(Object)

`look_at/1` is called whenever the user looks at `Object`. The default definition in `defaultRules.pl` will print `LongName` from `description/4`, followed by a newline.

1.1.5 title(Title)

`Title` contains the title of the game.

1.1.6 words(Object, List)

`words/2` indicates that the words in `List` will describe `Object`, which is used in identying the object the player is referring to. An object will be considered to match the description if the terms the player has typed are a subset of the terms in `List`

1.2 Describing Locations

1.2.1 adjacent(Loc1, Loc2, Dir1, Dir2)

`adjacent/4` asserts that there is a path from `Loc1` to `Loc2` in direction `Dir1`, and one from `Loc2` to `Loc1` in direction `Dir2`, that can always be used to travel.

1.2.2 building(Object)

`building/1` succeeds if `Object` is described as a `building` in `description/4`. Buildings are locations that are the exteriors of buildings.

1.2.3 connected(Loc1, Loc2, Dir, Pred)

`connected/4` indicates that there is a path from `Loc1` to `Loc2` in direction `Dir`, and that when the player attempts to travel it, `call(Pred)` should be executed, and the player should only succeed if that call succeeds. This is determined from `adjacent/4` and `connection/4`.

1.2.4 connection(Loc1, Loc2, Dir, Pred)

`connection/4` asserts that there is a path from `Loc1` to `Loc2` in direction `Dir`, and that when the player attempts to travel it, `call(Pred)` should be executed, and the player should only succeed if that call succeeds.

1.2.4.1 checkBook

`checkBook/0` is checked when the player attempts to leave the library; if they are carrying a book that has not been checked out, it will result in failure.

1.2.4.2 checkBungee

`checkBungee/0` is checked when the player attempts to jump off the swinging bridge; it only succeeds if the player possesses the bungee cord.

1.2.4.3 checkCard

checkCard/0 is checked when the player attempts to enter Lottie, and checks if the player is carrying and id card.

1.2.4.4 checkCardLair

checkCardLair/0 is checked when the player attempts to go through the door with the card swiper in the path to the lair.

1.2.4.5 checkDoor(Door)

checkDoor/1 is checked when the player attempts to enter a door that has the potential to be locked; it will succeed if the door is unlocked or fail if it is locked, and print the appropriate message either way.

1.2.4.6 checkCricket

checkCricket/0 is checked when the player attempts to pass the secret service, and only succeeds if the player is carrying a cricket.

1.2.4.7 checkFood

checkFood/0 is checked when the player attempts to pass the troll, and only succeeds if there is food located in the troll's location.

1.2.4.8 checkSS

checkSS/0 is checked when the player attempts to go past the president's secret service.

1.2.4.9 checkStairs

checkStairs/0 is checked when the player attempts to go up or down in Frey, and asks the player whether they would like to take the elevator or stairs. Taking the elevator results in a failure.

1.2.4.10 leaveDoor(Door)

leaveDoor/1 is checked when the player attempts to leave through a door that has the potential to be locked; it always succeeds and merely prints some text based on whether the door is locked or not.

1.2.5 dark(Location)

dark/1 asserts that Location is dark and that the player will need a light source to navigate through it.

1.2.6 fakeDir(Loc, Dir, Message)

fakeDir/3 allows the player to attempt to go in direction Dir from Loc, but prints Message instead of actually allowing the movement. This would be useful for cases of travelling to places that are not defined, such as insides of buildings.

1.2.7 fasttravel(Location)

`fasttravel/1` asserts that `Location` is a valid target for the user to bike to (assuming they have already visited it).

1.2.8 maze(Object)

`maze/1` succeeds if `Object` is described as a `maze` in `description/4`. Maze areas are part of a complicated path intended to be confusing.

1.2.9 place(Object)

`place/1` succeeds if `Object` is described as a `place` in `description/4`. Places are locations of some variety that are outside–apparently they can be dug in, according to a comment in `knowledgeBase.pl`, but being an important generic outside location that is not a building seems to be the criterion.

1.2.10 room(Object)

`room/1` succeeds if `Object` is described as a `room` in `description/4`. Rooms are indoor locations that are, well, rooms.

1.2.11 spot(Object)

`spot/1` succeeds if `Object` is described as a `building`, `place`, `room`, or `waypoint` in `description/4`.

1.2.12 waypoint(Object)

`waypoint/1` succeeds if `Object` is described as a `waypoint` in `description/4`. Waypoints are locations that are outside that are not particularly noteworthy in their own right, but rather are a path between some other, more important locations.

1.3 Describing Objects

Defining a new object should be fairly simple; only a relatively few number of predicates need to be defined. First, `description/4` should be defined to provide messages for the player to `look` at the object and give the game a common name to refer to it; Second, `words/2` should be defined to declare what the player can use to refer to it; Third, `pick_up/1` should be declared to define the behavior if the player attempts to pick it up. (Note, however, that for people and doors a default failure predicate for `pick_up/1` is defined). For objects that cannot be moved or taken by the player, this is all that is needed. For objects that can be moved, however, `see_object/1` should also be declared to define the message that should be printed when the player enters a location with it, and `put_down/1` should be declared to define the behavior of the player attempting to drop the object.

1.3.1 action(Command, Object)

`action/2` is used to define custom commands for specific objects; `Command` is a list of one or two words that are a command typed by the player that would not be otherwise recognized, and `Object` is the object it will be called on. This should probably only be used in cases where it only makes sense to perform a certain action on one item.

1.3.2 can_see(Thing)

can_see/1 succeeds if the player is in the same location as Thing.

1.3.3 common_name(Object, NoArticle, Article)

common_name/3 isn't actually really used for anything any more, since we changed the short names in description/4 to lower-case; perhaps it will be changed to a predicate to indicate the proper article at some point.

1.3.4 dead_thing(Object)

dead_thing/1 succeeds if Object is described as dead in description/4.

1.3.5 door(Object)

door/1 succeeds if Object is described as a door in description/4. Doors are objects specifically created to be able to be locked or unlocked, and not picked up.

1.3.6 food(Object)

food/1 indicates that Object is some sort of food.

1.3.7 key(Object)

key/1 succeeds if Object is described as a key in description/4. Keys are objects specifically created for locking or unlocking doors.

1.3.8 object(Object)

object/1 succeeds if Object is described as a object or fakeobject in description/4, or is a door, key, lightsource, or stuffed animal. Objects are generic things that can generally be picked up, put down, or looked at.

1.3.9 pick_up(Object)

pick_up/1 will be called any time the player attempts to pick up Object. If the predicate succeeds, the player will successfully pick up Object, otherwise it will remain at its current location. This predicate should always be defined for an object, since the player should always at least receive a message for attempting to pick up an object. For fake objects that cannot be picked up, this predicate should print a message and then fail.

1.3.10 put_down(Object)

put_down/1 will be called any time the player attempts to drop Object. If the predicate succeeds, the player will successfully drop Object, otherwise it will remain in the player's inventory. Either way, some sort of message should usually be printed (which should end with a newline). This predicate does not need to be defined for objects that cannot be picked up.

1.3.11 see_object(Object)

see_object/1 will be called any time the description of the location of the player is printed; it will be called for each object in that location. If the object is immobile and mentioned in the description of the location, this predicate should be defined; generally this predicate should only contain a writefw/2 statement (which should end with a newline) and perhaps some conditions.

1.3.12 stuffedanimal(Object)

stuffedanimal/1 succeeds if Object is described as a sa in description/4.

1.3.13 takeable(Object)

takeable/1 succeeds if Object is described as an object, key, sa, or lightsource in description/4.

1.3.14 weapon(Object)

weapon/1 indicates that Object is some sort of weapon.

1.3.15 writeable(Object, Description, Write, Erase)

writeable/4 indicates that Object is a writeable surface that the player can write on and erase. Description is the name of the object, Write is the instrument the player uses to write on it, and Erase is what the player uses to erase it.

1.4 Describing People

1.4.1 enter(Person)

enter/1 will be called any time the player enters the same location as Person.

1.4.1.1 animal_thrown(Animal, Index)

animal_thrown/2 a helper predicate for when the stuffed animal thrower throws a stuffed animal at the player; Index represents the player's choice of what to do.

1.4.2 person(Object)

person/1 succeeds if Object is described as a person in description/4.

1.4.3 picks_up(Person, List)

picks_up/2 defines a list of objects that Person will pick up if they are in a room that they enter.

1.4.4 see_person(Person)

Similar to see_object/1, see_person/1 will be called any time the description of the location of the player is printed; it will be called for each person in that location. Generally this predicate should only contain a writefw/2 statement (which should end with a newline) and perhaps some conditions.

1.4.5 wanders(Person, List)

wanders/2 defines the locations that Person can wander to.

1.5 Describing Monsters

1.5.1 monster(Object)

monster/1 succeeds if Object is described as a monster in description/4. Monsters are interactive things that generally hinder the player's progress, and are more generic than people, but are otherwise similar to people.

1.5.2 see_monster(Monster)

Similar to see_object/1, see_monster/1 will be called any time the description of the location of the player is printed; it will be called for each monster in that location. Generally this predicate should only contain a writefw/2 statement (which should end with a newline) and perhaps some conditions.

1.6 Describing Stores

Defining a store merely requires a description/4 predicate, an initial/2 predicate to define the location, and a wares/2 predicate to define what is for sale. Not defining a words/2 will mean the user cannot refer to it, and subsequently a pick_up/1 will not be needed either.

1.6.1 wares(Store, Wares)

wares/2 defines a list of pairs that Store sells, where each pair has the item as a key and the cost as the value.

Chapter 2

Commands

2.1 Managing Player Commands

2.1.1 `die`

`die/0` kills the player.

2.1.2 `display_exits(Place)`

`display_exits/1` displays the list of exits the player can take from `Place` using either `list_connections/1` or `list_exits/1` based on whether `showall` is asserted.

2.1.3 `do(CommandList)`

`do/1` handles the player typing in `CommandList`, obtained from `get_command/1`.

2.1.4 `inventory`

`inventory/0` prints the list of objects the player is carrying as well as how much money the player has.

2.1.5 `list_exits(Place)`

`list_exits/1` prints the list of directions the player can travel from `Place`.

2.1.6 `list_have`

`list_have/0` prints the list of items the player is carrying.

2.1.7 `list_monsters(Place)`

`list_monsters/1` prints the list of objects the player can see at `Place` by calling `see_monster/1`.

2.1.8 `list_people(Place)`

`list_people/1` prints the list of objects the player can see at `Place` by calling `see_person/1`.

2.1.9 list_things(Place)

list_things/1 prints the list of objects the player can see at Place by calling see_object/1.

2.1.10 look

look/0 prints the player's current location, the objects, monsters, and people they see, and the exits available.

2.1.11 move(Thing, Location)

move/2 retracts the old location/2 predicate for Thing and asserts a new one at Location. This is used for both objects and people.

2.1.12 print_help

print_help/0 displays the help messages.

2.1.13 show_money

show_money/0 prints a message displaying the amount of money the player currently has.

2.2 Navigating the Player

2.2.1 can_dir(Person, Direction, Place)

can_dir/3 called whenever a character is attempting to move in direction Direction; if successful, Place is unified with the new location Person would be in.

2.2.2 dir(ShortName, LongName)

dir/2 indicates that the abbreviation ShortName is associated with the full direction LongName.

2.2.3 fast_travel

fast_travel/0 is called whenever the player attempts to bike, and handles selecting location as well as actually moving the player.

2.2.4 go(Direction)

go/1 attempts to move the player in direction Direction.

2.2.5 go_dir(Person, Direction)

go_dir/2 attempts to move Person in direction Direction. Only used for the protagonist.

2.2.6 wait

wait/0 updates all non-player characters without moving the player.

2.3 Interacting with Locations

2.3.1 dig(Location)

`dig/1` is checked when the player attempts to dig; the location they are currently in is unified with `Location`, and so code to be executed when the player digs should be located in a `dig/1` predicate.

A default `dig/1` is defined in `defaultRules.pl` that tells the player they cannot dig in their current location.

2.3.2 jump

`jump/0` is called whenever the player attempts to jump.

2.3.3 lie_down

`lie_down/0` is called whenever the player attempts to lie down.

2.3.4 run

`run/0` is called whenever the player attempts to run.

2.3.5 try_buy

`try_buy/0` handles the player entering the `buy` command.

2.3.6 try_dig

`try_dig/0` handles the player entering the `dig` command.

2.4 Interacting with Objects

2.4.1 canLock(Key, Door)

`canLock/2` indicates that the player will successfully be able to lock `Door` if they posses `Key`.

2.4.2 canUnlock(Key, Door)

`canUnlock/2` indicates that the player will successfully be able to unlock `Door` if they posses `Key`.

2.4.3 drop(Thing)

`drop/1` attempts for the player to drop `Thing`.

2.4.4 eat(Food)

`eat/1` is checked whenever the player attempts to eat something. The default rule (in `defaultRules.pl`) is to print a message indicating the player canot eat it and fail. If the predicate succeeds, `Food` is moved to `nowhere` so the player does not have it in their inventory.

2.4.5 eat_description(Description)

`eat_description/1` determines what object the player is referring to by `Description` and attempts to eat it.

2.4.6 kick(Thing)

kick/1 is called whenever the player attempts to kick something.

2.4.7 kick_description(Description)

kick_description/1 determines what object the player is referring to by Description and attempts to kick it.

2.4.8 light_source

light_source/0 succeeds if the player has a functioning light source and fails otherwise.

2.4.9 lock(Door)

lock/1 handles the player attempting to lock Door.

2.4.10 lock_description(Description)

lock_description/1 determines what object the player is referring to by Description and attempts to nlock it.

2.4.11 look_description(Description)

look_description/1 determines what object the player is referring to by Description and looks at it.

2.4.12 put_description(Description)

put_description/1 determines what object the player is referring to by Description and attempts to drop it.

2.4.13 take(Thing)

take/1 attempts for the player to take Thing.

2.4.14 take_description(Description)

take_description/1 determines what object the player is referring to by Description and attempts to take it.

2.4.15 talk_to_description(Description)

talk_to_description/1 determines what person or monster the player is referring to by Description and attempts to talk to them.

2.4.16 try_eat(Object)

try_eat/1 handles the player attempting to eat Object.

2.4.17 unlock(Door)

unlock/1 handles the player attempting to unlock Door.

2.4.18 unlock_description(Description)

unlock_description/1 determines what object the player is referring to by Description and attempts to unlock it.

2.5 Interacting with People

2.5.1 give(Person, Thing)

give/1 is called whenever the player attempts to give Thing to Person. If it succeeds, Thing is moved to Person. The default rule is to print an appropriate message and fail.

2.5.2 give_description(PDesc, TDesc)

give_description/2 determines what object the player is referring to by TDesc, what person the player is referring to by PDesc, and attempts to give the object to the player.

2.5.3 kill(Person, Thing)

kill/1 is called whenever the player attempts to attack or kill Person with Thing. If it succeeds, the person (or monster) is replaced with a dead version. The default rule is to print an appropriate message and fail.

2.5.4 kill_description(PDesc, TDesc)

kill_description/2 determines what object the player is referring to by TDesc, what person the player is referring to by PDesc, and attempts to kill the person with the object.

2.5.5 talk_to(Person)

talk_to/1 is called whenever the player tries to talk to Person. Note that there are two default predicates for this in defaultRules.pl.

2.5.5.1 random_statements(Person, List)

random_statements/2 contains a list of statements that, if defined, will be chosen from randomly by a default talk_to/1 each time the player talks to Person.

2.5.5.2 sequential_statements(Person, List)

sequential_statements/2 contains a list of statements that, if defined, will be iterated through by a default talk_to/1 each time the player talks to Person.

2.5.5.3 talk_message(Index, List, Person)

talk_message/3 is merely a helper predicate for defining the default talk_to/1s for sequential and random statements; it will print the appropriate message from List

2.5.6 throw_description(PDesc, TDesc)

throw_description/2 determines what object the player is referring to by TDesc, what person the player is referring to by PDesc, and attempts to throw the object at the person.

2.5.7 toss(Person, Thing)

toss/1 is called whenever the player attempts to throw Thing at Person. The default rule is to print an appropriate message and fail.

2.6 Interacting with Stores

2.6.1 buy(Store)

buy/1 is called whenever the user attempts to buy something at a store, and will list the available wares and have the user select one.

2.6.1.1 attempt_purchase(Item, A)

attempt_purchase/2 is called once the player has selected an item to buy, and will check if the player has enough money, and if so transfer the item from the store to the player.

2.6.1.2 name_and_cost(A, Thing, String)

name_and_cost/3 is a helper method for use with select_thing/3 to list the available wares at a store.

2.6.2 purchased(Item)

purchased/1 is called whenever the player buys an item. If it fails, the object cannot be purchased. The default is to do nothing and succeed.

2.7 Controlling Game Flow

2.7.1 attempt_quit

attempt_quit/0 quits the game.

2.7.2 begin

begin/0 initializes everything for the game and displays the initial message.

2.7.3 command_loop

command_loop/0 controls the main loop of running the game.

2.7.4 continue_game

continue_game/0 prints a message welcoming the player back and begins the loop controlling the game.

2.7.5 display_ending

display_ending/0 prints the message for the game ending, based upon which ending the player has triggered.

2.7.6 end_win

`end_win/0` asserts `state(game_over)` so that the player is not allowed to perform actions.

2.7.7 start_game

`start_game/0` starts the game for the first time, and being running it.

2.7.8 restart

`restart/0` restarts the game, including re-printing the initial message.

2.8 Debug Commands

2.8.1 acquire_description(Description)

`acquire_description/1` determines what object the player is referring to by `Description` and magically acquires it.

2.8.2 list_connections(Place)

`list_connections/1` prints the list of places the player can travel to from `Place`, along with their associated directions.

2.8.3 list_people_locations

`list_people_locations/0` prints a list of characters and their current locations.

2.8.4 show_counters

`show_counters/0` prints a list of all counters.

2.8.5 warp(Place)

`warp/1` teleports the player to `Place`.

Part II

Standalone Files

Chapter 3

Getting Keypresses (keypress.pl)

3.1 External Predicates

These are the predicates that would be public if it were a module, and that are to be used outside of `keypress.pl`.

3.1.1 `get_char_set(List, Index)`

`get_char_set/2` waits for the user to enter a character in a list of lists of character codes in `List`, and unifies `Index` with the index of the list containing the entered code.

Example: `get_char_set(["Aa", "Bb"], X).`

This will wait for the user to enter an 'A' or 'B', and unify `X` with `0` for 'A' and `1` for 'B'.

3.1.2 `get_from_list(List, Index)`

`get_from_list/2` will print a list of options from character codes in `List`, wait for the user to select one, and unify `Index` with the index of the selection, or `-1` if the user cancels selection.

3.1.3 `get_YN`

`get_YN/0` will wait for the user to enter a 'Y' (in which case it succeeds) or an 'N' (in which case it fails.)

3.1.4 `select_thing(List, Pred, Thing)`

`select_thing/3` will take a list of things, unify each one with `call(Pred, Item, String)`, print a list of the strings obtained in this way, wait for the user to select one, and unify the selection with `Thing`, or unify `Thing` with `[]` if the user cancels.

3.2 Internal Predicates

These are the predicates that would not be public if it were a module, and should generally only be used inside of `keypress.pl`.

3.2.1 `get_list_index(Length, Index)`

`get_list_index/2` will wait for the user to enter a letter for a list selection, and unify `Index` with the index of the selection, or with `-1` if the user cancels.

3.2.2 `list_things(Index, List)`

`list_things/2` will print a list of things for a user selection, including a `[Z] Cancel` option.

Chapter 4

People Wandering (peopleCode.pl)

4.1 External Predicates

4.1.1 `initialize_wander_counters`

`initialize_wander_counters/0` This predicate should be called to initialize all of the counters for wandering people.

4.1.2 `update_moves`

`update_moves/0` will update all of the moves that have been indicated should be performed with `will_move/3`, check for wandering characters, print the appropriate messages, and run the appropriate code for interactions.

4.1.3 `will_move(Person, From, To)`

`will_move/3` will add `Person` to the list of people to moved, and will relocate `Person` from `From` to `To` (and print the appropriate messages) the next time that `update_moves/0` is called.

4.2 Internal Predicates

4.2.1 `check_wander`

`check_wander/0` checks each person, decrements their counter until they should wander, checks if it is `0`, and if so calls `wander_person/1` on them and resets their wander counter.

4.2.2 `clear_movements`

`clear_movements/0` retracts all clauses of `just_moved/3` in order to prepare for the next movement step.

4.2.3 `encounter`

`encounter/0` takes care of calling `enter` for every person who has a reaction to moving into the same location as the player.

4.2.4 pick_up_stuff(Person)

`pick_up_stuff/1` checks that `Person` will pick up anything in the room they are in that is defined in their `picks_up/2` predicate.

4.2.5 pickups

`pickups/0` checks all people and calls `pick_up_stuff/1` with each of them.

4.2.6 update_locations

`update_locations/0` updates the location of each person who needs to be moved.

4.2.7 wander(Person, Places, Dest)

`wander/3` picks a location that `Person` can wander to, updates `will_move/3`, and binds `Dest` to the location chosen.

4.2.8 wander_person(Person)

`wander_person/1` checks to see if `Person` should wander, and calls `wander/3` appropriately if so.

4.2.9 wandered_messages

`wandered_messages/0` prints the messages for people moving into and out of the player's location.

Chapter 5

Identifying Objects (identificationCode.pl)

5.1 External Predicates

5.1.1 identify(Object, Description, Pred)

identify/3 will attempt to find objects that fit the list of terms bound to Description and satisfy the predicate Pred; it will bind one at a time to Object until it fails when it cannot find any more.

5.1.2 get_object(Object, Description, Pred, None, Many)

get_object/5 will bind Object to the unique object satisfying identify/3, fails and prints None if none are found, or presents the user with a selection after printing Many if multiple objects would succeed.

5.1.3 get_object(Object, Description, Pred, NoneEmpty, NoneWords, Many)

get_object/6 will bind Object to the unique object satisfying identify/3, fails and prints NoneEmpty if none are found and no description was given, fails and prints NoneWords if none are found a description was provided, or presents the user with a selection after printing Many if multiple objects would succeed.

5.1.4 get_objects(List, Object, Description, Pred)

get_objects/4 will bind List to the list of all objects that would be returned by identify/3.

5.2 Internal Predicates

5.2.1 choose(Object, List, Description, None, Many)

choose/5 will take the list of items obtained by get_objects/4, print the None message if none are obtained, print the Many message and prompt the user with a selection if multiple objects fit the description, and bind Object with the appropriate item, or fail if there was either no appropriate item or the user cancelled selection.

Chapter 6

Getting Commands from the User (getcommand.pl)

6.1 External Predicates

6.1.1 `get_command(List)`

`get_command/1` will print a prompt, read a line of input, strip of non-alpha-numeric characters, and split into a list of pairs of terms which are bound to `List`.
Each line that the user enters is split by prepositions and returned as a list of pairs, the first pair having a key of the first word and a value of the non-preposition terms following it, and each subsequent pair beginning with a preposition and followed by the list of terms. Articles (a, an, the) are ignored.
Examples:

"Go to the dark room" would become `[go-[], to-[dark, room]]`

"Kill the troll with the nasty knife" would become `[kill-[troll], with-[nasty, knife]]`.

6.2 Internal Predicates

6.2.1 `descriptor(List, Term)`

`descriptor/2` takes a difference list `List` and binds `Term` to the list of words from the beginning of `List` that are not prepositions (this would be the sequence of words that will be treated as a description for an object in the game).

6.2.2 `expand(List, OldList)`

`expand/2` will expand abbreviations in `Oldlist` and unify it with `List`.

6.2.3 `piece(List, TermList)`

`piece/2` takes a difference list `List` and recursively binds `TermList` to the list of preposition-descriptor pairs in the list.

6.2.4 prep(List, Term)

`prep/2` takes a difference list `List` and binds `Term` to the first word if it is a preposition, and fails otherwise.

6.2.5 prepp(List, Pair)

`prepp/2` takes a difference list `List` and binds `Pair` to a preposition-descriptor pair if `List` begins with a preposition.

6.2.6 preposition(Word)

`preposition/1` indicates that `Word` should be parsed as a preposition.

6.2.7 splitList(List, PairList)

`splitList/2` takes a list of words `List` and converts it to a list of pairs as would be output by `get_command/1` and binds it to `PairList`.

6.2.8 strip_downcase(In, Out)

`strip_downcase/2` will remove all non-space, non-alphanumeric characters from `In` and unify it with `Out`.

6.2.9 strip_irrelevant(List, OldList)

`strip_irrelevant/2` will remove all article terms from `Oldlist` and unify it with `List`.

6.2.10 verb(List, Term)

`verb/2` takes a difference list `List` and binds `Term` to the first word, which shall be treated as the verb in the command.

6.2.11 verbp(List, Pair)

`verbp/2` takes a difference list `List` and binds `Pair` to a verb-descriptor pair where the verb is the first element of `List`.

Chapter 7

Forcing Output to Wrap at Words (writewrap.pl)

7.1 External Predicates

7.1.1 `writefw(Format)`

Equivalent to `writef(Format, [])`.

7.1.2 `writefw(Format, Arguments)`

`writefw/2` works similarly to `writef/2` except that it will check the terminal width and ensure that line breaks occur at a space character.

7.2 Internal Predicates

7.2.1 `wrapline(Line, Size)`

`wrapline/2` splits the given `Line` into words seperated by spaces and prints them using `write_pieces/4`.

7.2.2 `wraplines(List, Size)`

`wraplines/2` recursively prints a list of lines, wrapping appopriately at length `Size`.

7.2.3 `write_piece(CurPos, Piece, NextPos, Size)`

`write_piece/4` will write one word, starting with a new line if the current position (`CurPos`) plus the length of the word is greater than `Size`, and binding `NextPos` to the new position on the line.

7.2.4 `write_pieces(CurPos, List, NextPos, Size)`

`write_pieces/4` recursively writes a list of words, wrapping as appopriate, using `write_piece/4`.

7.2.5 `write_space(CurPos, NextPos, Size)`

`write_space/3` will write a space, or a newline instead if the position at `CurPos` would instead wrap onto a new line (of length `Size`), and binds `NextPos` to the new position on the line.

7.2.6 `writewrap(Format, Arguments, Size)`

`writewrap/3` writes similarly to `writef/2`, but wraps at the end of a line of length `Size` only at spaces.

Part III

Dynamic Predicates

Chapter 8

Dynamic Predicates

8.1 showall

`showall/0` is a debug feature that, if asserted, indicates that the locations in each directions should be printed with the `look` command.

8.2 counter(Counter, X)

`counter/2` indicates that `Counter` is set at `X`.

8.3 dead(Thing)

`dead/1` indicates that `Thing` is dead. Currently only relevant for `prot`.

8.4 just_moved(Person, From, To)

`just_moved/3` indicates that `Person` will be moved the next time `update_moves/0` is called.

8.5 location(Thing, Place)

`location/2` indicates that `Thing` is in `Place`. This could be the location of a person, the location of an object, or the location of the player (`prot`).

8.6 locked(Door)

`locked/1` indicates that `Door` is locked.

8.7 money(Amount)

`money/1` indicates that the player has `Amount` units of currency.

8.8 state(Thing)

`state/1` is used for generic things that need to be set to track something.

8.9 unchecked(Thing)

`unchecked/1` indicates that `Thing` has not been checked out at the library. Currently this is only relevant for `importantBook`.

8.10 unlocked(Door)

`unlocked/1` indicates that `Door` is unlocked.

Chapter 9

Working with Dynamic Predicates

9.1 Initialization

9.1.1 `init_locked(Door)`

`init_locked/1` indicates that `locked(Door)` should initially be asserted.

9.1.2 `init_objs`

`init_objs/0` will initialize all dynamic predicates when called.

9.1.3 `init_state(State)`

`init_state/1` indicates that `state(State)` should intially be asserted.

9.1.4 `initial(Thing, Location)`

`initial/2` indicates that `location(Thing, Location)` should initially be asserted.

9.2 Working with Counters

9.2.1 `decrement_counter(Thing)`

`decrement_counter/1` will decrement the counter of `Thing`, or set it to `-1` if it has not been previously defined.

9.2.2 `increment_counter(Thing)`

`increment_counter/1` will increment the counter of `Thing`, or set it to `1` if it has not been previously defined. It will also check if any events need to happen as a result of that incrementation.

9.2.3 `set_counter(Thing, Value)`

`set_counter/2` will set the counter of `Thing` to `Value`, retracting the previous value if it exists.

9.3 Counting Money

9.3.1 `add_money(Amount)`

`add_money/1` will add `Amount` to the amount of money the player has, or set it to `Amount` if the player's money has not been initialized.

9.3.2 `sub_money(Amount)`

`sub_money/1` will subtract `Amount` from the amount of money the player has, or set it to `-Amount` if the player's money has not been initialized, though this should probably not be allowed to happen– after all, this is a player and not a government.

9.4 State

9.4.1 `set_state(X)`

`set_state/1` will assert `state(X)` unless it has already been asserted.