Ergo Documentation

Version 1.0

Delay Desforêts

janv. 21, 2019

Table des matières:

Ceci est la page d'accueil de la documentation du jeu Ergo.

CHAPITRE 1

Wiki

1.1 Présentation du jeu

Le point de départ est le jeu « Ergo » : The_Game_of_Proving_You_Exist Vous pouvez retrouver les règles détaillées ici

Le jeu est composé de 55 cartes : 4 de chaque variable (A, B, C ou D), 4 de chaque opérateur (ET, OU, =>), 6 cartes NON, 8 parenthèses, 3 cartes Ergo et 10 cartes particulières.

Chaque joueur (4 maximum) se voit assigné une variable (A, B, C ou D) au début du jeu. À chaque manche, les joueurs essaient collectivement de créer une preuve de leur existence tout en réfutant l'existence des autres joueurs. À chaque tour, un joueur pioche deux cartes et doit jouer deux cartes (éventuellement les défausser). Lorsque une carte Ergo est jouée ou qu'il n'y a plus de carte dans la pioche la preuve est terminée. À condition qu'il n'y ait pas de paradoxe, chaque joueur dont l'existence est prouvée reçoit un nombre de points égal au nombre de cartes dans la preuve. Toutes les cartes sont ensuite mélangée et une nouvelle manche est lancée. Le premier joueur ayant 50 points gagne.

Concernant la construction de la preuve, un certain nombre de règles doivent être respectées :

- la preuve doit avoir au maximum 4 lignes. Dès qu'elle atteint 4 lignes, toutes les cartes supplémentaires doivent être jouées sur une de ces lignes ;
- chaque ligne doit être syntaxiquement correcte (deux opérateurs ou deux variables ne peuvent pas se suivre, chaque parenthèse ouvrante doit correspondre à une parenthèse fermante, dots) ;
- une carte peut être insérée entre deux cartes déjà posées à condition que le résultat reste syntaxiquement correct.

1.2 Le projet

Le but est de réaliser une implémentation en Python de ce jeu. Pour cela, je vois plusieurs points à traiter, plus ou moins par ordre de difficulté croissante :

- analyser une ligne de la preuve pour vérifier qu'elle est syntaxiquement correcte ;
- coder une ligne syntaxiquement correcte sous une forme exploitable (arbre, forme conjonctive normale, forme disjonctive normale, … ?) ;
- déterminer à partir du codage des 4 lignes quelles variables sont prouvées ou s'il y a une contradiction ;
- réaliser une interface graphique (à priori avec tkinter) ;

— implémenter une fonction pour pouvoir jouer contre l'ordinateur.

Si on finit tout ça et qu'on a peur de s'ennuyer, on pourra toujours creuser pour améliorer la façon dont l'ordinateur joue. Au pire, on demandera à Frédéric Muller de nous prêter ses TetrisBot pour qu'ils apprennent à jouer à Ergo ;-)

CHAPITRE 2

Documentation des modules

2.1 Module cards

Gestion des cartes et des preuves.

class cards.**Card**(*name*)

Les cartes du jeu.

__init__(*name*)

Constructeur de la classe

Paramètres **name** (*string*) – nom de la carte (ET, OU, AND, NOT, . . .)

Retourne Objet Card

Type retourné *Card*

__repr__()

Retourne le nom de la carte.

Type retourné string

__weakref__

list of weak references to the object (if defined)

is_close()

Indique si la carte est une parenthèse fermante.

Type retourné boolean

is_ergo()

Indique si la carte est « Ergo ».

Type retourné boolean

is_letter()

Indique si la carte est une lettre ou non.

Type retourné boolean

is_not()

Indique si la carte est un « NOT ».

Type retourné boolean

is_open()
Indique si la carte est une parenthèse ouvrante.

Type retourné boolean

is_operator()
Indique si la carte est un opérateur.

Type retourné boolean

priority()

Retourne le niveau de priorité de la carte. Si la carte n'a pas de niveau de priorité, lève une exception.

Type retourné int

turn_parenthesis()
Retourne la parenthèse, si c'en est une, ne fait rien sinon.

class cards.**CardList**(**args*)
Liste de cartes, avec la Notation Polonaise Inversée associée. Si la liste ne correspond pas à une preuve syntaxiquement correcte, NPI vaut None.

__init__(**args*)
Constructeur de la classe.

Paramètres **args** – des arguments pour construire la liste

Retourne objet CardList initalisé avec les éléments passés par args

Type retourné *CardList*

__weakref__
list of weak references to the object (if defined)

append(*card*)
Ajoute la carte card à la fin de la liste.

Paramètres **card** (Card) – la carte à ajouter

evalue(*interpretation*)
Évalue la liste en fonction du modèle. npi doit être calculé. :param interpretation : liste de 4 booléens correspondant aux valeurs de A, B, C et D

Retourne Valeur de la liste de carte en fonction du modèle.

Type retourné boolean

insert(*index*, *card*)
Insère card à la position index.

Paramètres
- **card** (Card) – la carte à insérer
- **index** (*int*) – la position à laquelle insérer la carte

is_syntactically_correct()
Indique si la liste de cartes est syntaxiquement correcte, sans s'occuper de la correspondance des parenthèses.

Retourne True si la liste est syntaxiquement correcte, False sinon

Type retourné boolean

pop(*index=-1*)
Supprime la carte en position index (par défaut la dernière) et la renvoie.

Paramètres **index** (*int*) – la position de la carte à renvoyer

Retourne la carte supprimée

Type retourné *Card*

to_npi()
Met à jour self.npi en :

— None si la syntaxe de la liste n'est pas correcte
— une liste de carte correspondant à la notation polonaise inversée de la liste de départ sinon.

class cards.**Deck**

Le paquet de cartes.

__init__()

Constructeur de la classe

Retourne un paquet de cartes mélangées

Type retourné *Deck*

__weakref__

list of weak references to the object (if defined)

append(*card*)

Ajoute une carte (possible avec Tabula Rasa).

Paramètres **card** (Card) – la carte à ajouter

draw(*number*)

Retourne number cartes du paquet s'il en reste assez, la fin du paquet ou une liste vide sinon.

Type retourné list

is_finished()

Indique si la paquet est terminé.

Retourne True si la paquet est terminé, False sinon

Type retourné boolean

reset()

Réinitialise le paquet de cartes.

class cards.**Proof**

Classe gérant les prémisses et la preuve.

__init__()

Initialisation des attributs :
— premises : liste de 4 CardList correspondant aux 4 lignes de prémisses ;
— currently_added : liste de cartes venant d'être ajoutées aux prémisses, et pas encore validées.

Retourne un objet Proof

Type retourné *Proof*

__weakref__

list of weak references to the object (if defined)

all_cards_played()

Indique si chacune des 4 cartes A, B, C et D a été jouée, et donc s'il est possible de jouer la carte Ergo.

Type retourné boolean

conclusion()

Retourne None si les prémisses conduisent à une contradiction, ou une liste associant à chaque variable "A", "B", "C" et "D" soit True si elle est prouvée, False si la négation est prouvée, on None si on ne peut rien conclure.

Type retourné list ou NoneType

insert(*premise*, *index*, *card*)

Insère la carte card dans la prémisse premise en position index et actualise currently_added.

Paramètres
— **premise** (*int*) – le numéro de la prémisse
— **index** (*int*) – la position à laquelle insérer la carte dans la prémisse
— **card** (Card) – la carte à insérer

Retourne True si l'insertion est possible, False sinon.

Type retourné boolean

is_all_correct()
Indique si toutes les prémisses sont correctes ou pas.

Type retourné boolean

pop(*premise*, *index*)
Enlève la carte en position index de la prémisse premise à condition qu'elle vienne d'être ajoutée.

Paramètres
— **premise** (*int*) – le numéro de la prémisse
— **index** (*int*) – la position dans la prémisse de la carte à supprimer

Retourne la carte en question ou None si on ne peut pas l'enlever.

Type retourné Card ou NoneType

reset_added()
Remise à zéro de currently_added pour le prochain tour.

score()

Retourne Le score correspondant à la preuve, c'est à dire le nombre de cartes qui la compose.

Type retourné int

2.2 Module gui

Interface graphique.

class gui.**ErgoGui**
Interface graphique.

__init__()
Constructeur de la classe

Retourne Objet ErgoGui

Type retourné *ErgoGui*

__init_canvas__()
Création du canvas de jeu avec les lignes des prémisses, les mains et noms des joueurs et la pile

__init_menu__()
creation de la barre de menu qui permet d'afficher l'aide, les règles, la version et de pouvoir quitter le jeu.

affiche_cards(*card_list*, *row*)
affiche la liste de carte card_list à la ligne row (0 à 3 pour les prémisses, 4 pour la main du joueur

Paramètres
— **card_list** (*list*) – la liste de cartes à afficher
— **row** (*int*) – le numéro de la ligne

display_current_player()
Affiche les numéros de joueurs en faisant tourner, le joueur courant est toujours en haut à gauche.

drop(*event*)
Place la carte marquée « selected » sur la grille, et l'ajoute au bon endroit (prémisse, main ou pile) et enlève la marque « selected ». Si c'est impossible, la remet à la fin de la main.

Paramètres **event** (*tkinter.Event*) – événement

fin_manche()
Fin de la manche, affichage des gagnants et du score.

move(*event*)
Déplace la carte marquée « selected ».

Paramètres **event** (*tkinter.Event*) – événement

play()
Valide un coup si possible, et passe au joueur suivant

quitter()
Quitte

rules()
Affiche les règles du jeu à partir du fichier regles_ergo.txt

select(*event*)
Selectionne une carte, la marque comme « selected », la met en avant plan, et l'enlève de l'endroit où elle était (mains, prémisse ou pile).

Paramètres **event** (*tkinter.Event*) – événement

switch(*event*)
Retourne la parenthèse si c'en est une, dans la main du joueur courant.

Paramètres **event** (*tkinter.Event*) – événement

version()
Affiche la version du jeu

CHAPITRE 3

Index et tables

— genindex
— modindex
— search

Index des modules Python

c

g

Index

T

V