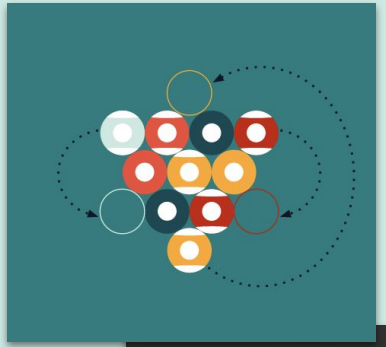
Soutenance Coding Weeks 2023
Simulation Physique

BILLARD INTERACTIF

Implémentation d'une interface graphique
et d'un moteur physique

Groupe 24
DELAVAUD Paul-Émile - FAYNOT Guillaume
MONTOYA Samuel - NOEL–BERTIN Paul - TALBAUT Gatien

GENÈSE DU PROJET



Kick-off

- Augmentation de la **difficulté** ⇒ Choix du billard
- Volonté de développer un **moteur physique**

Découpage en objectifs, sprints, fonctionnalités

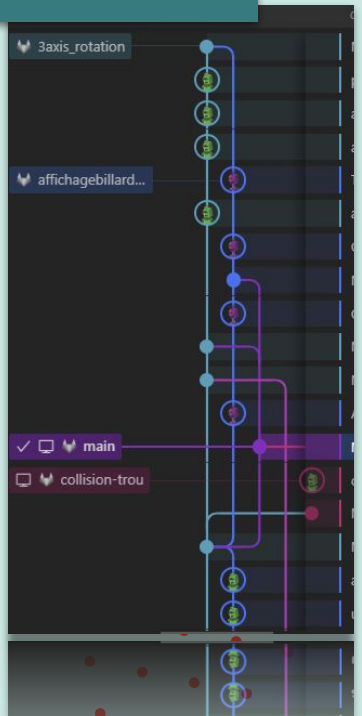
- Conception d'un **MVP**
- Développement de fonctionnalités en plus

Utilisation de Git

- Branche **main** verrouillée par un modérateur
- 1 fonctionnalité = 1 branche
- Commits **réguliers**, descriptions **précises**, relecture après **merge request**
- Utilisation de fichiers **.gitignore**

README.md + WorkingDocs

- **requirements.txt** et instructions de lancement
- Explication de la **structure** du projet + suivi des tests



MVP

ANALYSE DU PROBLÈME ET DES PRINCIPALES FONCTIONNALITÉS MVP

- Permettre à l'utilisateur d'**interagir avec le billard** et **régler un tir** : choix de la direction et de la puissance du coup
- Permettre la **simulation** et la **visualisation** du coup (en 2D vue du dessus) en fonction des paramètres renseignés par l'utilisateur : **déplacements**, **rebonds**, **collisions**
- **Afficher** la simulation sous la forme d'une animation à l'aide du module **matplotlib**

MVP

FONCTIONNALITÉS SUPPLÉMENTAIRES

- Modéliser des **frottements** (résistance au roulement et au glissement)
- Modélisation de la **rotation** sur 3 axes des boules
- Ajouter des **trous** à la table de billard
- Intégrer la possibilité de **jouer** une partie de billard **français/américain/anglais**

UN PEU DE PHYSIQUE ...

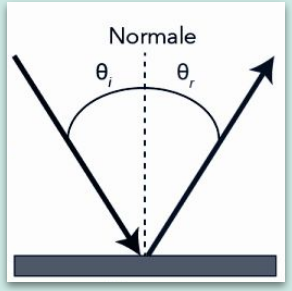
COLLISIONS

Conservation de la quantité de mouvement et énergie cinétique

$$\frac{\mathrm{d}\vec{p}}{\mathrm{d}t} = \vec{0}.$$

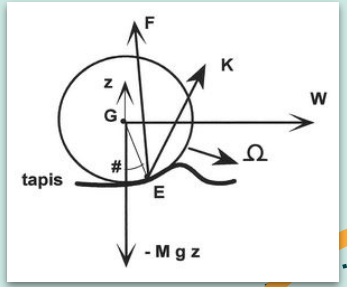
REBONDS

Lois de l'optique



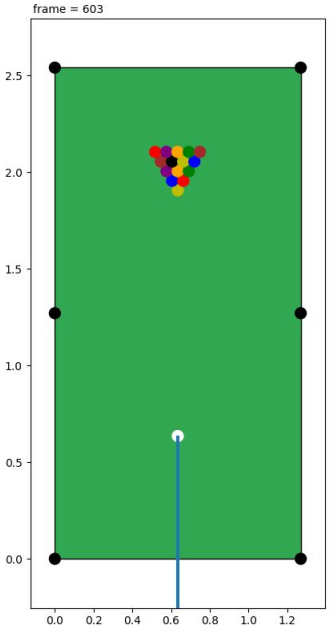
FROTTEMENTS

Resistance au roulement



OBJECTIF I :

Création d'un billard opérationnel sans GUI



MVP : ORGANISATION EN SPRINT

MODÉLISATION DU BILLARD

MODÉLISATION DU BILLARD

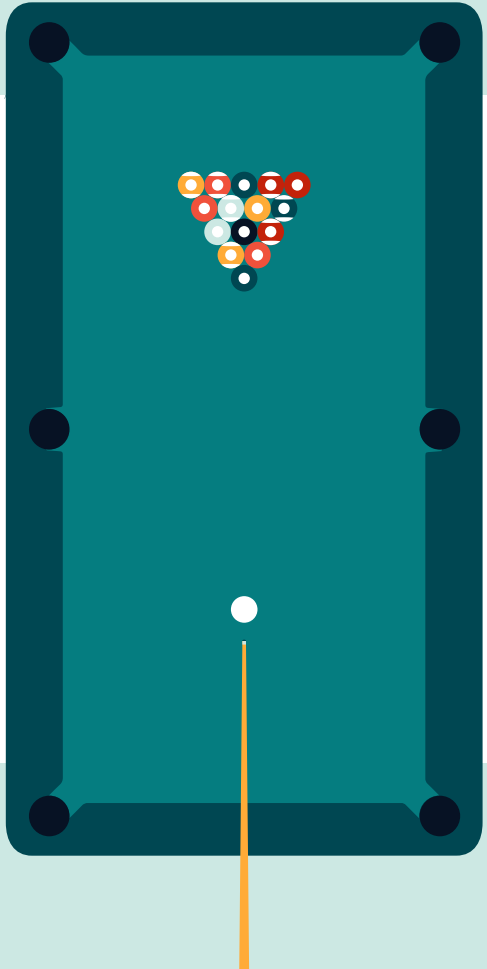
OBJETS ET AFFICHAGE

1) Représentation des objets :

- *classes Pool, Board, Ball et Cue*

2) Affichage de ces objets avec matplotlib :

- *fonction “trace”*



OBJET_GAME.PY

CUE()

- *mass, energy, angle*

- frappe()
- update_angle()

BALL()

- *number, initial_position, radius, mass, state, color*

- set_size()
- update_position()
- update_speed()
- update_state()

BOARD()

- *length, width*

- get_corners()
- get_pockets()
- get_middle()
- set_size()

POOL()

- *number_of_balls*
- *1 instance de BOARD*
- *1 dictionnaire contenant number_of_balls BALL*

GRAPHIQUE.PY

TRACE()

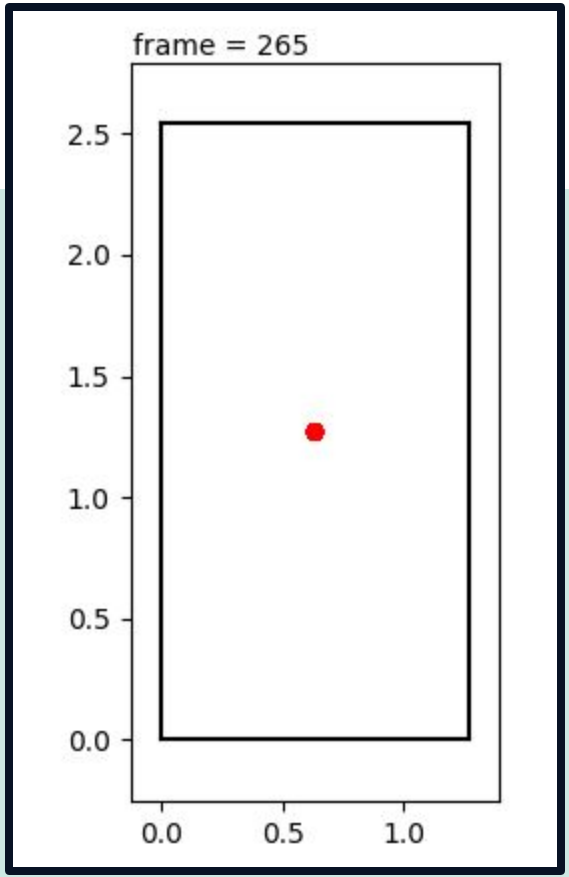
- *billard, dynamic_func*
- return fig, ani

TRACE.UPDATE()

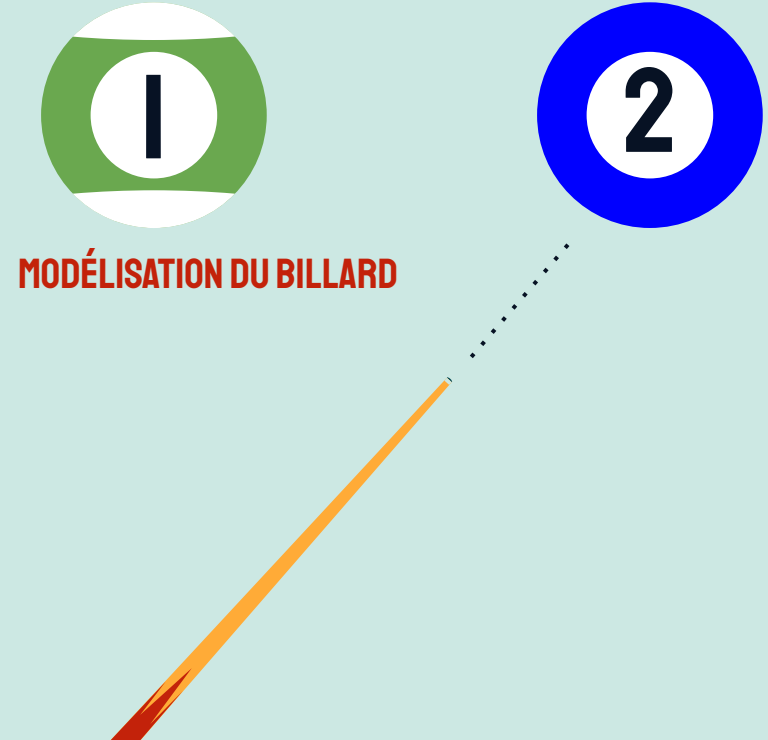
- *frame*
- return circles, frame_text, rectangle

Cf. slide 14

SPRINT I



MVP : ORGANISATION EN SPRINT



MODÉLISATION DU BILLARD

IMPLÉMENTATION DES MOUVEMENTS LIBRES

IMPLÉMENTATION DES MOUVEMENTS LIBRES

3) Simulation d'un **coup** :

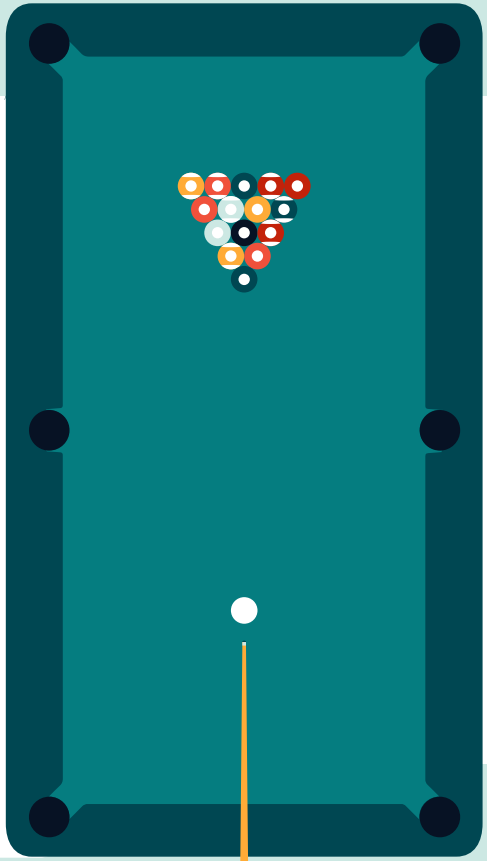
- *méthode frappe() dans la classe Cue*

4) Implémentation des **rebonds** :

- *fonction de détection de rebond*
- *calcul des nouvelles positions et vitesses*

5) Mise à jour des boules :

- *fonction "update(billard,Δt)" dans graphique.py*



DYNAMIC.PY

DETECT()

- *board, ball, delta_t*
- **return** bounce_status

un tuple si ball est en dehors de board après dt

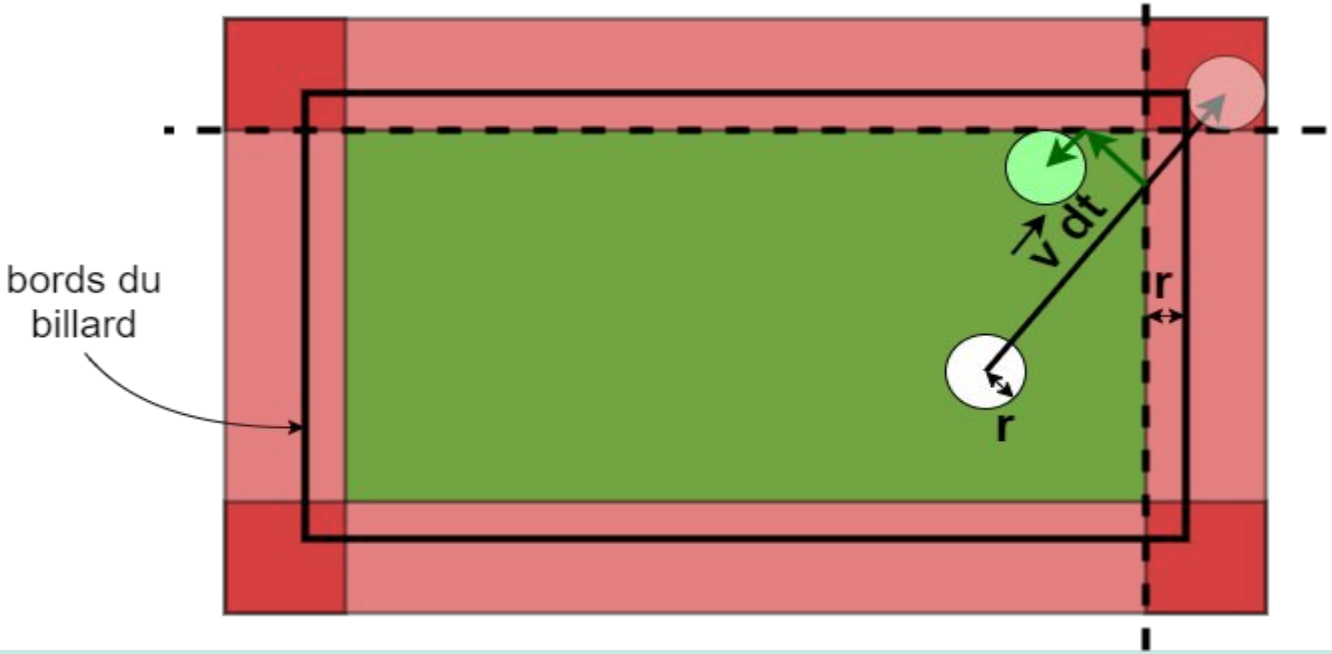
REBOND()

- *board, ball, delta_t, bounce_status*
- **return** pos_reel, speed_reel

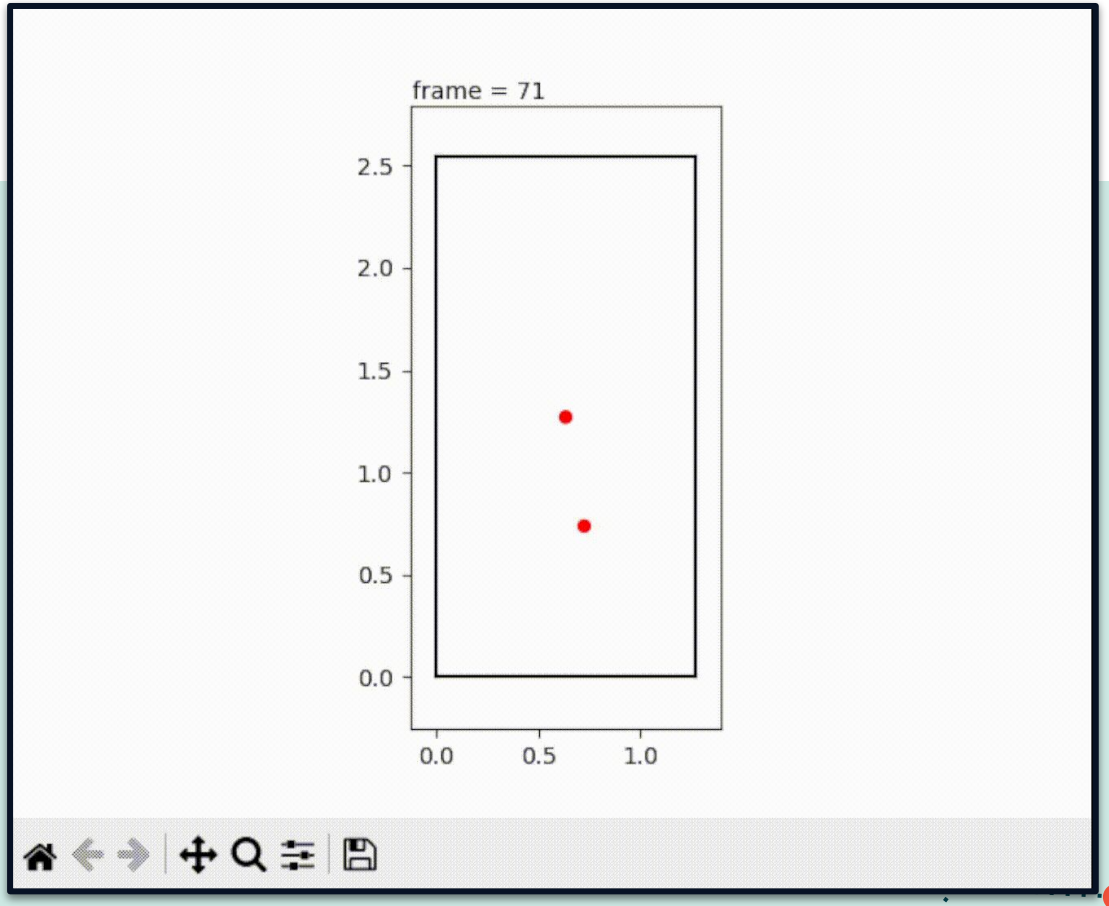
UPDATE_POOL()

- *pool, delta_t*
- Itération sur les éléments de pool.balls
  - detect()
  - rebond()
  - ball.update_speed()
  - ball.update_position()

SCHEMA DE PRINCIPE DU REBOND



SPRINT 2



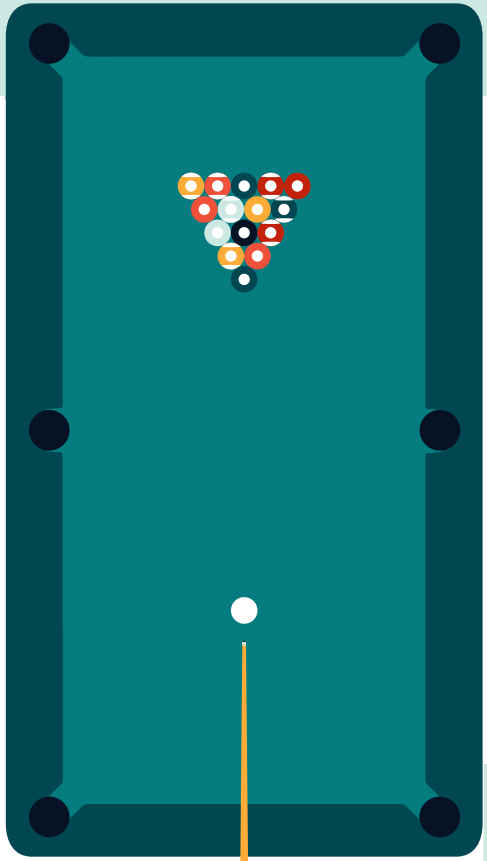
MVP : ORGANISATION EN SPRINT

MODÉLISATION DU BILLARD

IMPLÉMENTATION DES MOUVEMENTS LIBRES

IMPLÉMENTATION DES INTERACTIONS PHYSIQUES

IMPLÉMENTATION DES INTERACTIONS PHYSIQUES



6) Modélisation de chocs élastiques entre les boules :

Hypothèse : pas de temps court ⇒ collision entre 2 boules

- *_ détection de choc (update naive)*
- *_ calcul du premier choc chronologiquement (repéré par t0)*
- *_ update(billard, t0)*
- *_ mise à jour des* vitesses et des positions après le choc

7) Modélisation des frottements avec le tapis. Mise à jour des vitesses sur chaque pas de temps Δt :

- *diminution d'un % α de la vitesse*
- *arrêt de la boule si sa vitesse est inférieure à* v_{min}

DYNAMIC.PY

COLLIDED()

- *ball1, ball2*
- **return** # un tuple si ball1 et ball2 sont en contact

COLLISION_MATRIX()

- *pool*
- **return** matrix

matrix[i][j]=True si collision, False sinon

IMPACT_TIME()

- *ball1, ball2*
- **return** # le temps avant impact de deux boules

DYNAMIC.PY

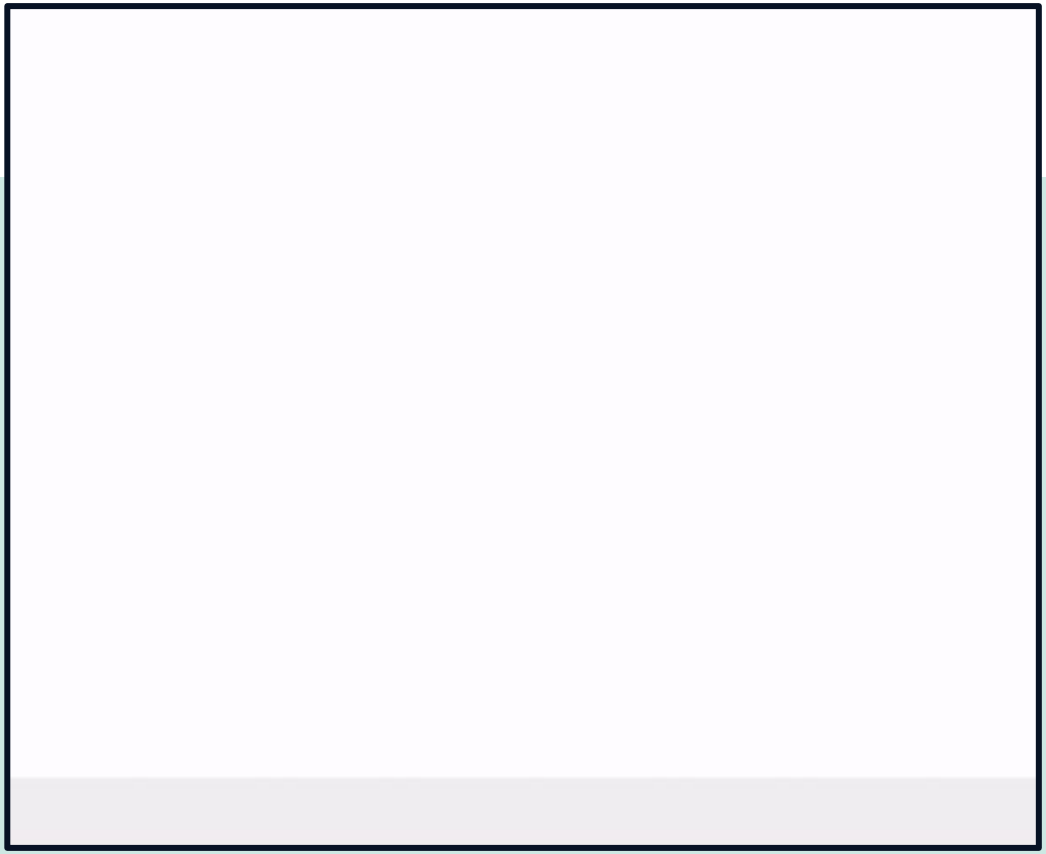
UPDATE_SPEED_2COLLIDEDBALLS()

- *pool, index*
- # met à jour les vitesses des deux boules (index) qui se sont choquées élastiquement en prenant en compte leur orientation et leur vitesse initiale.

FIRST_IMPACT()

- *pool, matrix*
- **return** # le temps avant le 1er impact, un tuple index contenant les deux numéros des boules qui sont en collision en premier

SPRINT 3



MVP : ORGANISATION EN SPRINT

MODÉLISATION DU BILLARD

IMPLÉMENTATION DES MOUVEMENTS LIBRES

IMPLÉMENTATION DES INTERACTIONS PHYSIQUE

PARAMÉTRISATION SANS GUI

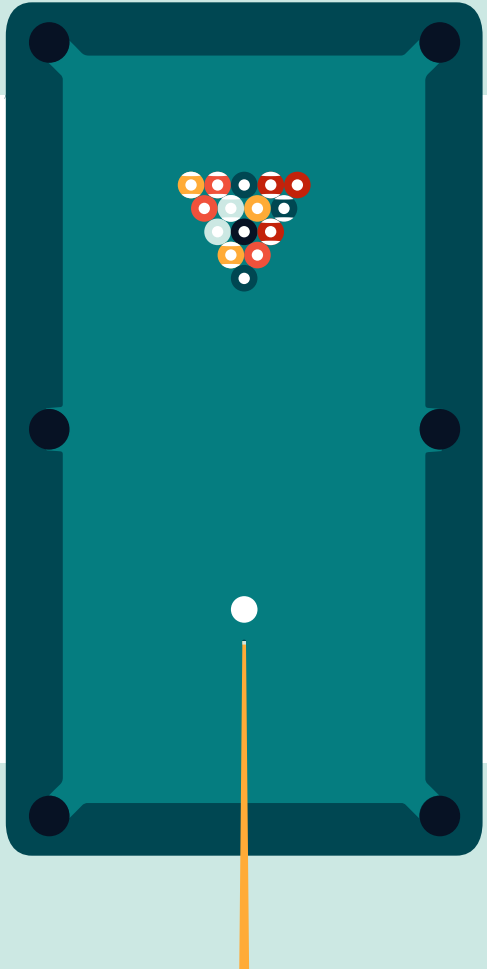
PARAMÉTRISATION SANS GUI

8) **Initialisation** de la position des boules:

- *mise à jour de la classe Pool selon le type de billard (américain, français ou anglais).*

9) Lancement d'un coup depuis **une invite de commande** :

- *utilisation du module "argparse"*



LAUNCH.PY

STRUCTURE

```
def main():
    ...
if __name__ =='__main__':
    main()
```

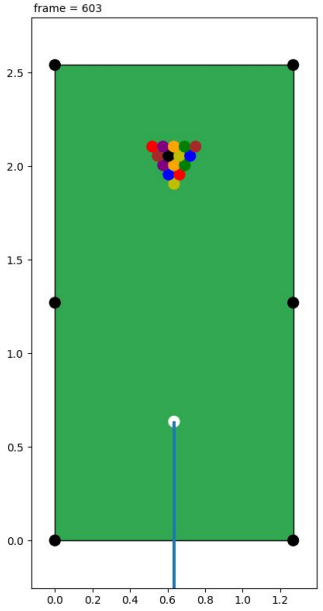
APPEL

Dans un terminal:

launch.py "number_of_balls", "mass", "energy", "angle", "AfficherAnimation"

OBJECTIF 2:

Création d’un billard interactif avec GUI



INTERFACE GRAPHIQUE UTILISATEUR (GUI)

1. INITIALISATION DE LA PARTIE
2. LANCEMENT D'UN COUP
3. GESTION DE LA PARTIE
4. AMÉLIORATIONS ESTHÉTIQUES

INITIALISATION DE LA PARTIE

INITIALISATION DE LA PARTIE

Sélection des modes de jeu (français, anglais ou américain). En fonction du choix, il faut :

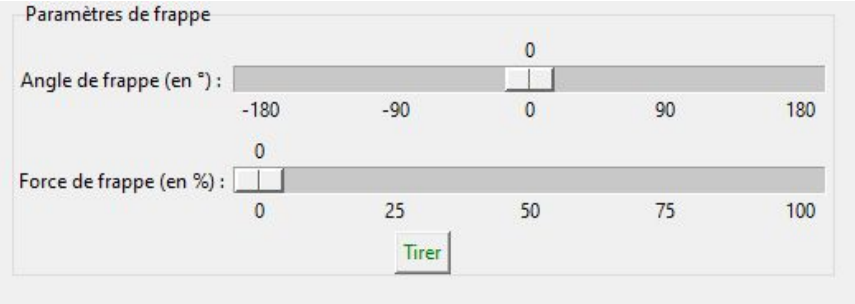
- Adapter les boules (nombre, dimension et position)
- Afficher ou non les trous
- Réinitialiser la queue

LANCEMENT D'UN COUP

LANCEMENT D'UN COUP

Permettre à l'utilisateur de jouer un coup et de sélectionner :

- La puissance
- L'angle

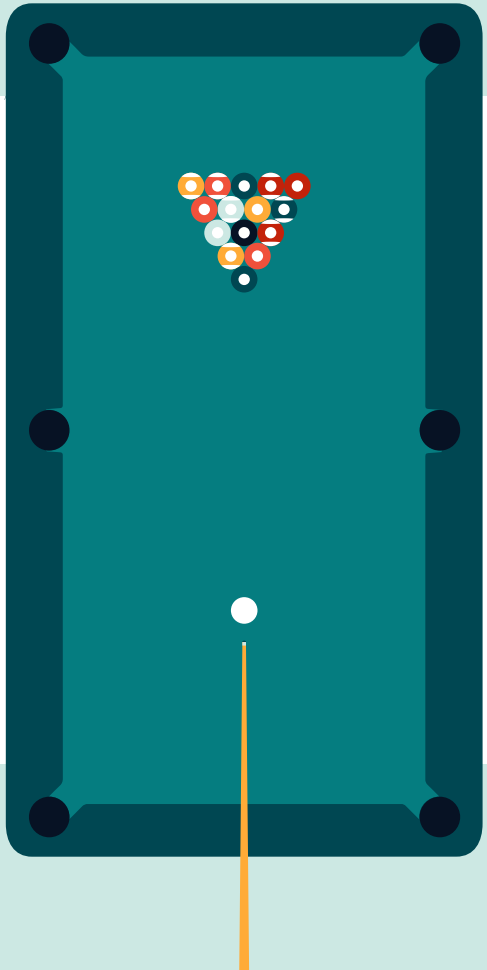


GESTION DE LA PARTIE

GESTION DE LA PARTIE

Assurer le bon déroulement de la partie.

- Bloquer les tirs quand nécessaire
- Mise hors-jeu de boules tombées dans les trous
- Gestion de la fin de partie



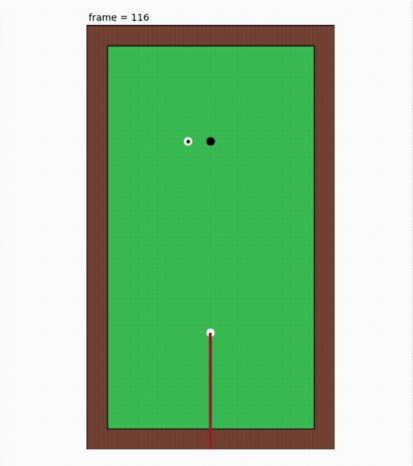
4

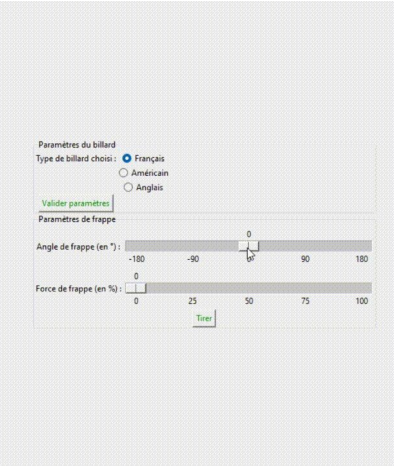
AMÉLIORATIONS ESTHÉTIQUES

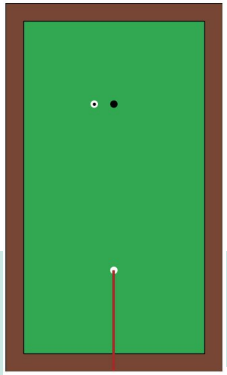
AMÉLIORATIONS ESTHÉTIQUES

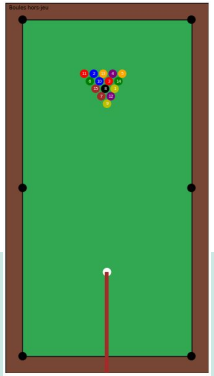
Proposer un produit final agréable.

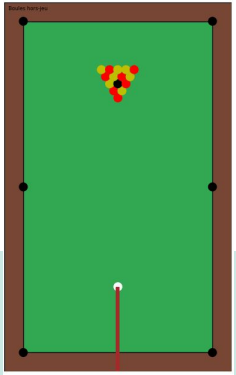
- Avoir un aspect fidèle à la réalité (boule, bords,...)
- Affichage fluide (compromis calcul/optimisation)









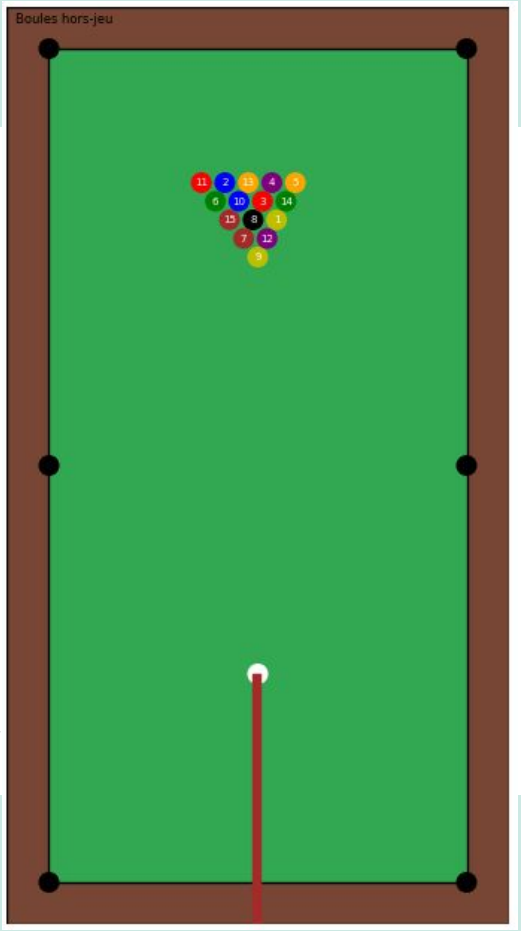


POUR ALLER PLUS LOIN

Autres fonctionnalités

OBJECTIF 3:

Ajout des rotations sur 3 axes



MODÉLISATION DE LA ROTATION SUR 3 AXES POUR LES EFFETS

1

ROTATION DANS LES AXES DU PLAN (X ET Y) SOUS PYMUNK

UTILISATION DE LA BIBLIOTHÈQUE PYMUNK POUR UN ROTATION SUR Z FONCTIONNELLE

2

ROTATION SUR TROIS AXES AVEC NOTRE "MOTEUR PHYSIQUE"

MODÉLISATION DE LA ROTATION SUR Z

PORTAGE DES FONCTIONNALITÉS PRÉCÉDENTES

1

ROTATION DANS LES AXES DU PLAN (X ET Y) SOUS PYMUNK

OBJET_PYMUNK.PY

BOARD()

- *pymunk.Segment*

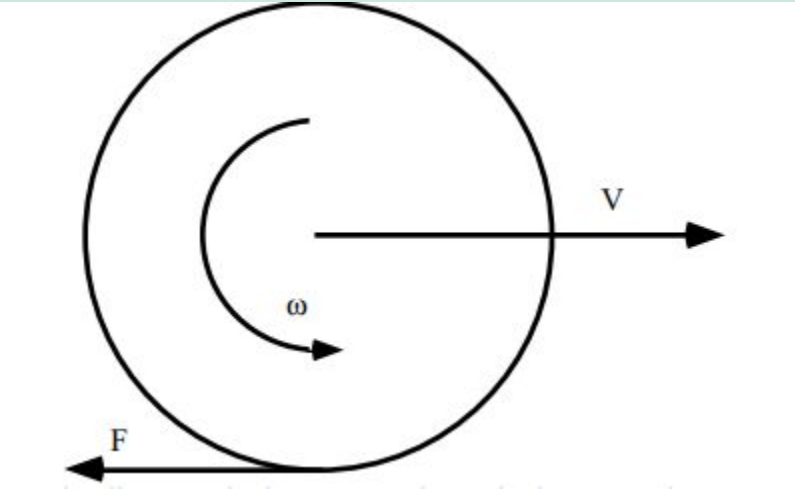
BALL()

- *angle_velocity_xy array (2,)*
- *pymunk.Body*
- *pymunk.Circle*
- *Surcharge de pymunk.Body.update_velocity()*

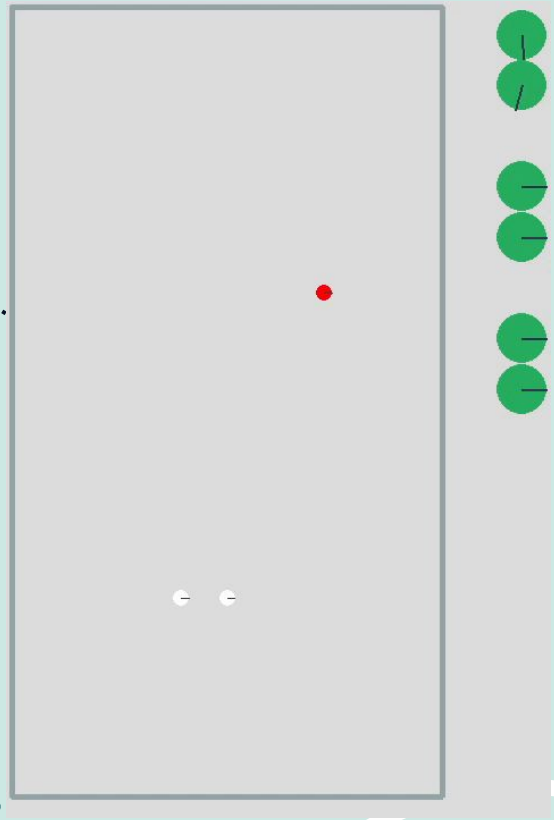
APP()

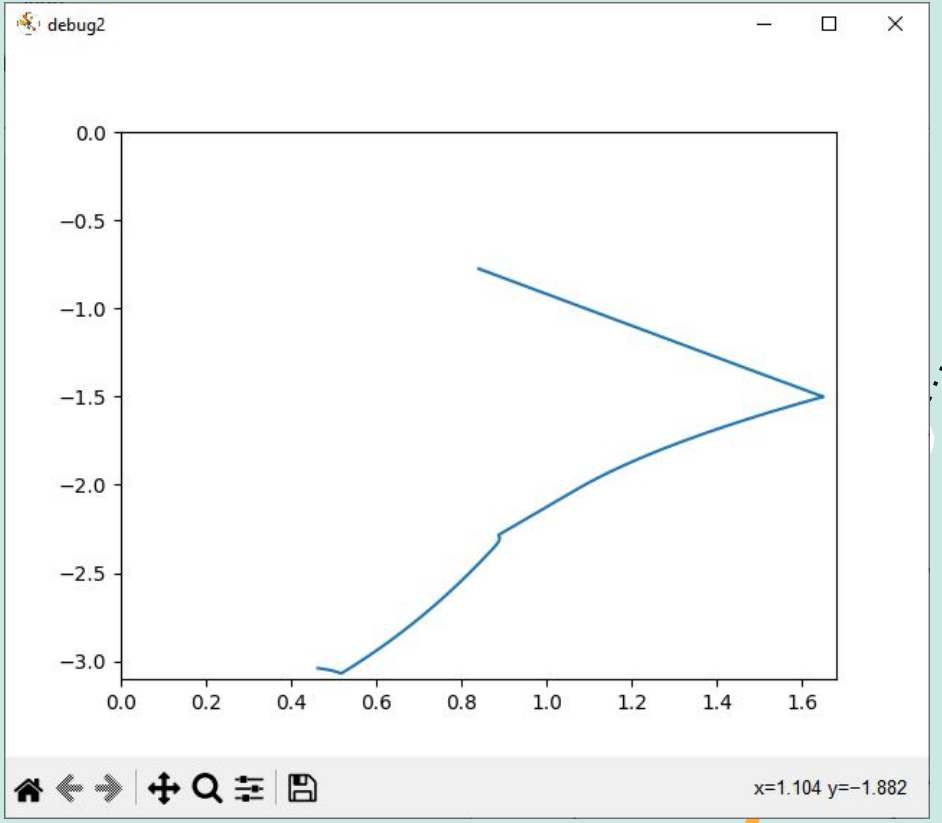
- *pymunk.Space, pool*
- Affichage avec *pymunk.space.debug_draw()* et mise à jour des positions avec *pymunk.space.step()*

Schéma de l’effort du tapis sur la boule dû au frottement



Amateur Physics for the Amateur Pool Player - Third Edition, Ron Shepard




debug2
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
x=1.104 y=−1.882

2

ROTATION SUR TROIS AXES AVEC NOTRE "MOTEUR PHYSIQUE"

OBJET.PY

BALL()

- *angle_velocity_xyz array (3,)*
- *friction_tapis*
- *resistance_roulement_tapis*
- *inertia*

- *apply_force()*
- *apply_torque()*

CUE()

- Modification de frappe() pour prendre en paramètre un position d'impact de la queue sur la boule

DYNAMIC.PY

APPLY_FRICTIONS()

- Application des forces de frottement force de résistance au roulement pour chaque boule

UPDATE_BALL_BOUNCE()

- Calcul de la vitesse de la boule après delta_t
- Changement de la vitesse lors d'un rebond en fonction de la rotation de la boule

UPDATE_POOL()

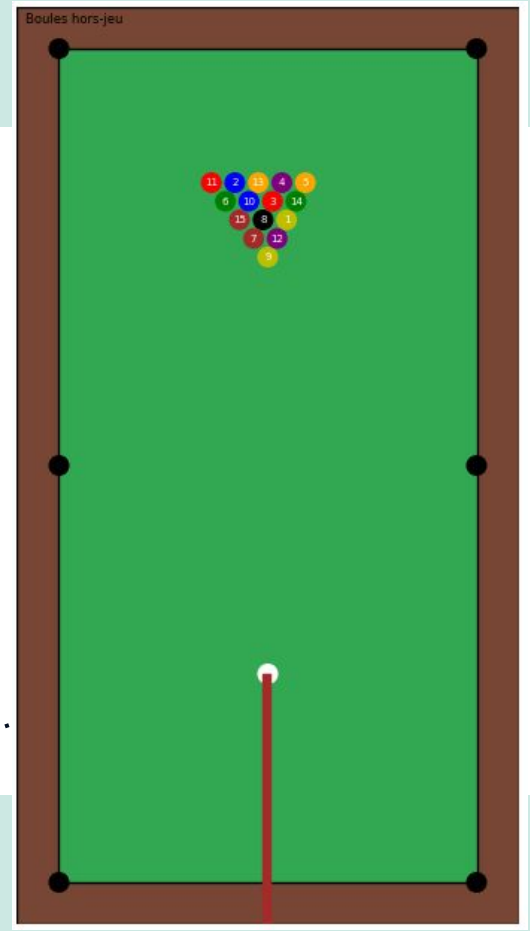
- Vérification des collisions
- Application des efforts pour modifiers les vitesses
- Mise à jour des positions

- return #booléen pour savoir si toutes les boules sont à l'arrêt (*at_equilibrium(pool)*)

TESTS

TDD

RED ⇒ GREEN ⇒ REFACTOR

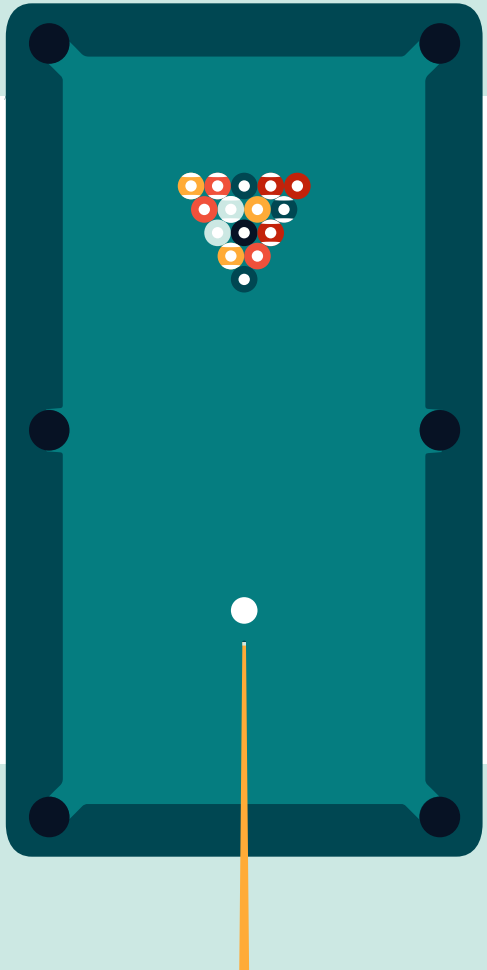


FONCTIONS DE TEST

- Appel du module pytest-cov

Appel dans le terminal :

```
pytest --cov=billard-cw --cov-report html
```



MERCI POUR VOTRE ATTENTION!