

# プロジェクト演習 テーマD

## 第2回

担当：CS学部 講師 伏見卓恭  
連絡先：[fushimity@edu.teu.ac.jp](mailto:fushimity@edu.teu.ac.jp)

# 授業の流れ

- 第1回:実験環境の構築 / Pygameの基礎 / Gitの基礎
- 第2回:Pygameによるゲーム開発の基礎 / コード規約とコードレビュー
- 第3回:オブジェクト指向によるゲーム開発 / GitHubの応用
- 第4回:Pygameによるゲーム開発の応用 / 共同開発の基礎
- 第5回:共同開発演習（個別実装）
- 第6回:共同開発演習（共同実装）
- 第7回:共同開発演習（成果発表）

# 本日のお品書き

## 1. 前回の復習

## 2. リーダブルコード

- コメント, docstring
- 型ヒント, 関数アノテーション
- コード規約

## 3. Pygameの演習

## 4. コードレビュー

目標：Pygameの理解を深め、読みやすいコードを実装でき、  
GitHubでIssueの送受信ができる

# 前回の復習

# 配布物の確認

- Moodleからex02.zipをDLし, ProjExD2023フォルダの下に配置

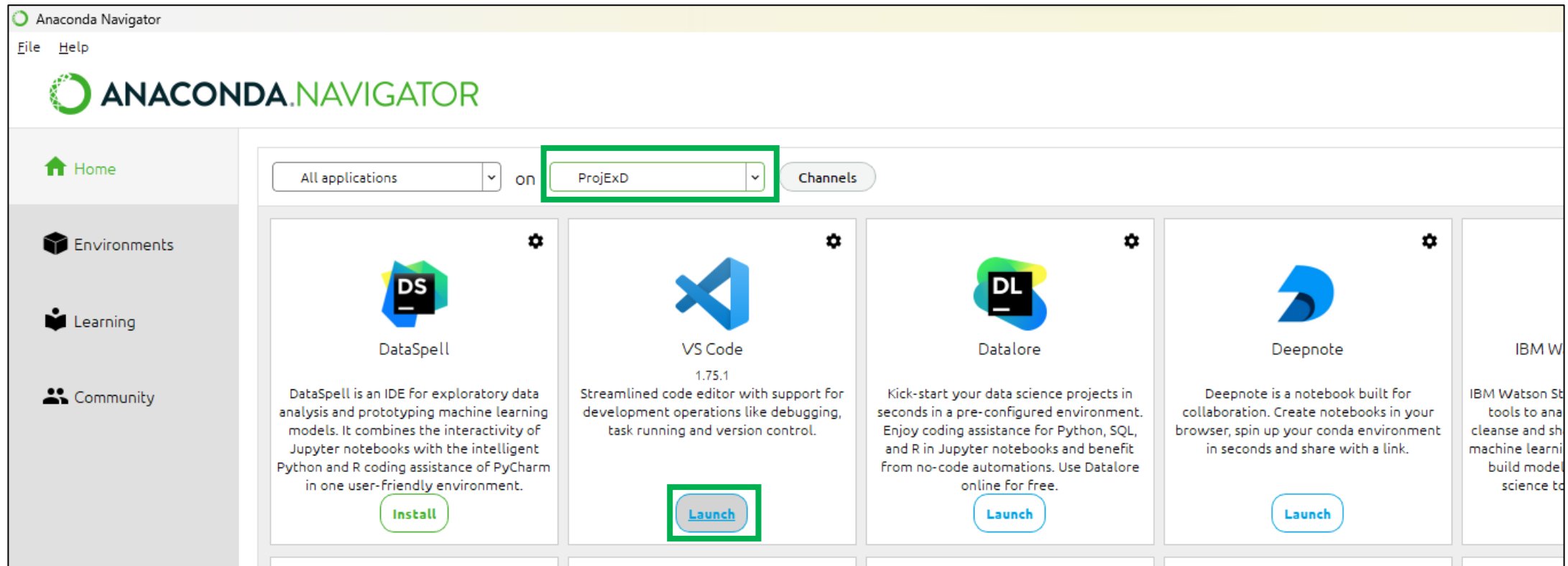
## 【配布物配置後のディレクトリ構造】

- ProjExD2023/
  - sample.py
  - ex01/
  - ex02/
    - dodge\_bomb.py . . . 逃げろ！こうかтон
    - fig/
      - pg\_bg.jpg . . . 背景画像
      - {0, ..., 9}.png . . . こうかтон画像

本日の配布物

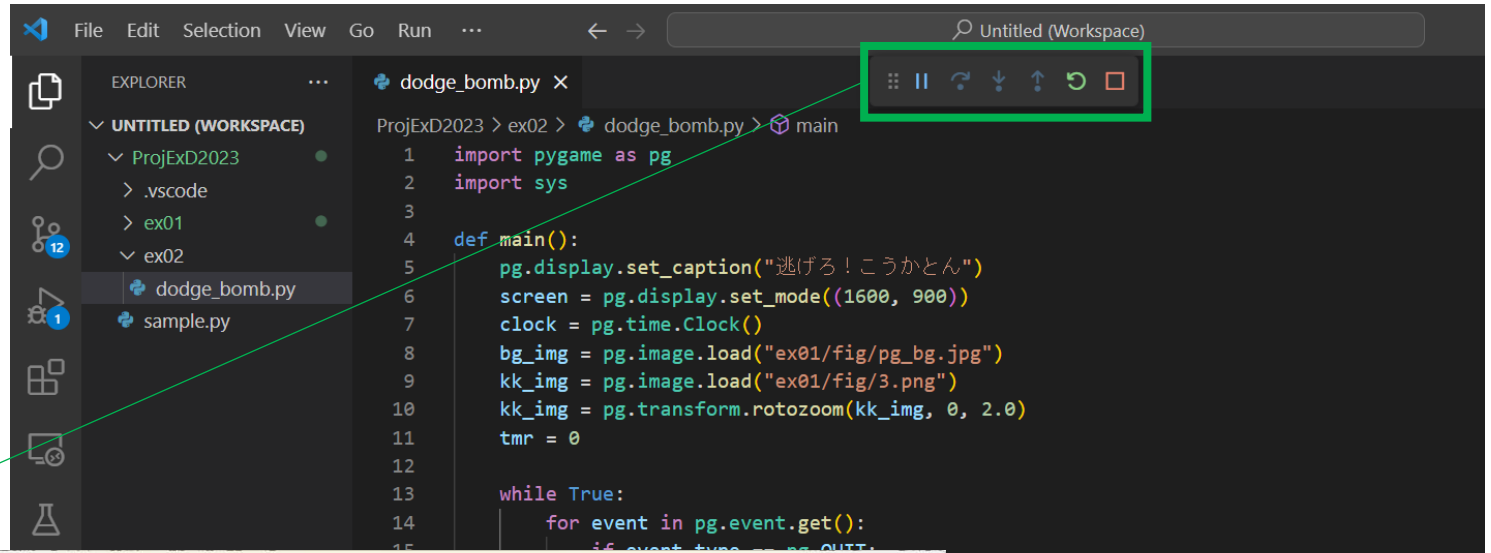
# VScodeの起動


- anaconda navigator → 「ProjExD」を選択 → VScodeのLaunch

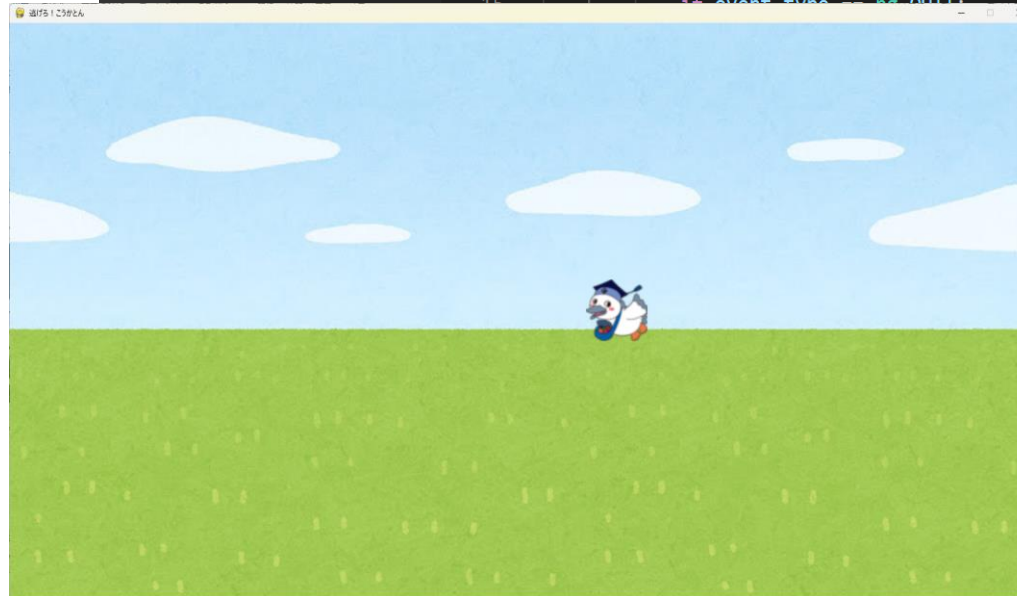


# まずは, dodge\_bomb.pyを動かしてみる

- 「Ctrl+S」で保存する
- 「Ctrl+F5」で実行する

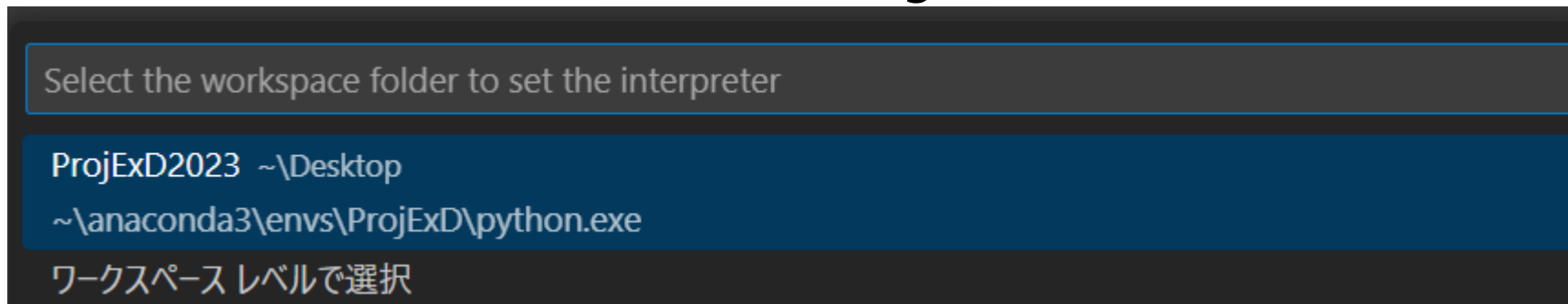


- 「」で終了する

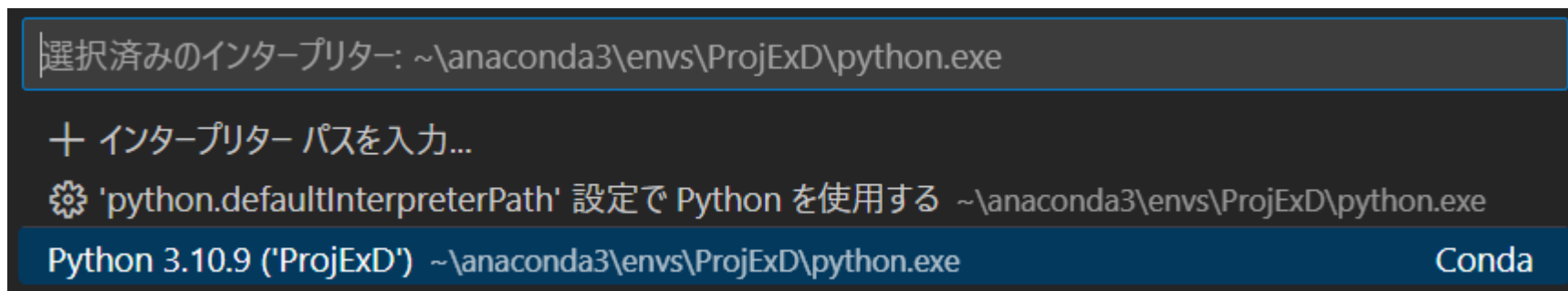


# 実行できなかったら → VScode右下を確認

- Python 3.10.9 ('ProjExD': conda) となっておらず,  
「Select...」またはPythonのバージョンだけになっている場合は,  
クリックして,
- ワークスペースフォルダ「ProjExD2023」を選ぶ



- インタプリタパス「Python3.10.9 ('ProjExD')」を選ぶ





# 前回の復習：dodge\_bomb.py

## 【ゲームの初期化】

- ウィンドウタイトルを「逃げろ！こうかとん」とする
- 幅1600×高さ900のスクリーンSurfaceを生成する
- 背景画像pg\_bg.jpgのSurfaceを生成する
- こうかとん画像3.pngを2倍に拡大したSurfaceを生成する

## 【ゲームのループ】

- 背景画像をスクリーンSurfaceに貼り付ける
- こうかとん画像をスクリーンSurfaceの横900，縦400に貼り付ける
- 画面を更新する

# 前回の復習：git

- git bashを起動し, ex02フォルダに移動する：`cd Desktop/ProjExD2023/ex02`
- ex02フォルダでgitリポジトリを初期化する：`git init`
- 全ファイルをステージングする：`git add *`
- 「初期状態」というコメントでコミットする：`git commit -m "初期状態"`
- GitHubに「ProjExD\_02」という公開リポジトリを作成する
- 公開リポジトリの情報を「origin」という名前で登録する：  
`git remote add origin https://github.com/fushimity/ProjExD_02.git`
- 公開リポジトリにコミット履歴をプッシュする：`git push origin main`

3限：リーダーダブルコード

# コメント

Pythonでは、複数行のコメントを書く構文はないが、ダブルクォーテーション3つで複数行の文字列を作ることができる。単なる文字列に対しては何もしないため、コメントとして機能する。

## コメント

```
print()
```

```
# 1行のコメント
```

```
"""
```

```
複数行の文字列  
複数行の文字列  
複数行の文字列
```

```
"""
```

- ← 「#」の後は半角スペース1つ入れる
- ← 行末の場合は「#」の前に半角スペース2つ入れる
- ← インデントの深さはコードと合わせる
- ← 長いコメントは行末（インラインコメント）ではなく、独立した行コメントにする
- ← 何をやっているか明らかなことは書かない

# docstring

モジュールやクラス、関数に関する説明の複数行コメントのことで、モジュールの先頭、クラス定義の直後、関数定義の直後に書かれる。docstringの内容は特殊属性`__doc__`に格納される。`help()`関数により、docstringの内容を確認することもできる。

docstring

```
def hoge():
```

```
    """
```

```
        複数行の文字列
```

```
        複数行の文字列
```

```
    """
```

```
print(hoge.__doc__)
```

```
def read_names(file_path: str):
```

```
    """
```

```
        poke_names.txtを読み込む関数
```

```
        引数：ファイルのパス
```

```
        戻り値：名前文字列のリスト、タイプリストのリスト、...
```

```
    """
```

```
print(read_names.__doc__)
```

← `help(read_names)`でもOK



```
read_names(file_path: str)
```

```
    poke_names.txtを読み込む関数
```

```
    引数：ファイルのパス
```

```
    戻り値：名前文字列のリスト、タイプリストのリスト、進化先リストのリスト
```

# 型ヒント, 関数アノテーション

- Pythonのオブジェクトは, 型(type), 値(value), 同一性(id)の3要素からなり, 型を明示的に示すことを型ヒントという
- 関数定義部で用いることが多い → 関数アノテーションと呼ぶ
- あくまでヒント, アノテーションであり, エラーを出したりはしない

## 型ヒントの例

# 宣言と代入を同時に行う場合

```
level: int = 51
```

明らかな場合は省略しても問題ない

# 宣言と代入を別々に行う場合

```
level: int
```

```
level = 51
```

## 関数アノテーションの例

```
def __init__(self, name: str, types: str) -> None:
```

# docstringと関数アノテーションの使用例

```
def check_bound(area: pg.Rect, obj: pg.Rect) -> tuple[bool, bool]
```

```
"""
```

```
オブジェクトが画面内か画面外かを判定し，真理値タプルを返す
```

```
引数1 area：画面SurfaceのRect
```

```
引数2 obj：オブジェクト（爆弾，こうかとん）SurfaceのRect
```

```
戻り値：横方向，縦方向のはみ出し判定結果（画面内：True／画面外：False）
```

```
"""
```

```
yoko, tate = True, True
```

```
if obj.left < area.left or area.right < obj.right: # 横方向のはみ出し判定
```

```
    yoko = False
```

```
if obj.top < area.top or area.bottom < obj.bottom: # 縦方向のはみ出し判定
```

```
    tate = False
```

```
return yoko, tate
```

docstringや  
関数アノテーション  
を書いておくと，

関数使用時に，VScodeが  
ヒントを表示してくれる

```
if check_boun
```

check\_bound

```
bb_rct.move_ip(vx, vy)
```

```
screen.blit(bb_img, bb_rct)
```

```
pg.display.update()
```

```
clock.tick(1000)
```

```
def check_bound(  
    area: Rect,  
    obj: Rect  
) -> tuple[bool, bool]
```

オブジェクトが画面内か画面外かを判定し，真理値タプルを返す 引数1 area：画面SurfaceのRect 引数2 obj：オブジェクト（爆弾，こうかとん）SurfaceのRect 戻り値：横方向，縦方向のはみ出し判定結果

# コード規約

- コード規約：<https://pep8-ja.readthedocs.io/ja/latest/>
- わかりやすい記事：  
<https://qiita.com/simonritchie/items/bb06a7521ae6560738a7>

**「一貫性にこだわりすぎるのは、狭い心の現れである」**  
つまり、規約に囚われすぎない臨機応変さも必要



# コード規約（つづき）

以下，抜粋

- 文中改行
  - 要素の先頭を合わせる
  - 1つインデントを入れる
  - 引数の場合は2つインデントを入れる
- 空行の数
  - トップレベルは2行入れる
  - それ以外は1行入れる
- import文
  - カンマで区切って1行にまとめず，1つずつ書く  
ただし，同じモジュールから複数の関数やクラスをimportするときは，1行にまとめる
  - 標準ライブラリ→サードパーティー→自作の順
  - 基本はアルファベット順

# コード規約（つづき）

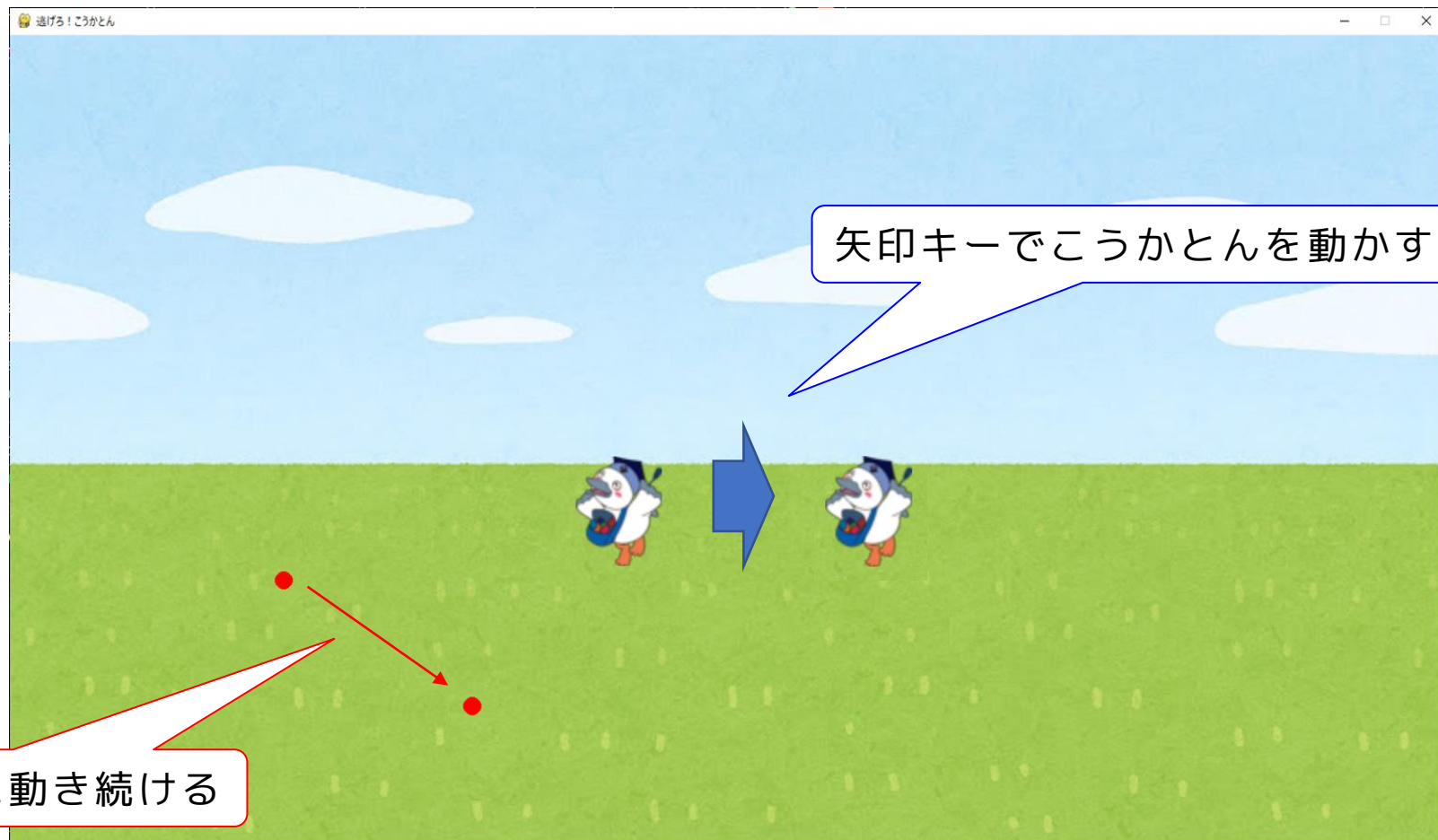
- 空白
  - 余分な空白は入れない
  - 統一する（カンマの後ろ，演算子の前後）
  - 「＝」の位置を合わせるために空白を入れない
  - キーワード引数，デフォルト引数の「＝」前後は空白を入れない
- 命名規則
  - 意味を表した単語にする（よく知られた省略は可能）
  - クラス名：パスカルケース（例：`SampleClass`）
  - 関数名，変数名：スネークケース（例：`sample_func`）
  - 定数名：全て大文字でアンダースコアでつなげる（例：`SAMPLE_CONST`）
  - 予約後などと被る場合：後ろにアンダースコアを付ける（例：`id_`）
- 条件文
  - Noneとの比較：「==」は使わず「is None」，「is not None」
  - 真理値との比較：「==」は使わない
    - N Gな例：`if key_lst[pg.K_UP] == True, if key_lst[pg.K_UP] != True`
    - O Kな例：`if key_lst[pg.K_UP], if not key_lst[pg.K_UP]`

# 練習問題（ Pygameの基本 ）

dodge\_bomb.py

# 「逃げろ！こうかとん」を実装しよう

- 野原で遊ぶこうかとんに爆弾が襲い掛かる。  
爆弾から逃げるゲームを実装する。



# 練習問題

※1問ずつステージング, コミットしよう

## 1. 爆弾Surfaceを作成し, ランダムな位置に配置せよ

- 半径: 10 / 色: 赤 の円
- 爆弾Surfaceの黒い部分を透明にするには「set\_colorkey(黒)」を利用する
- 爆弾Rectの位置を表す変数に乱数を設定する
- ためしにwhileループの中でblitして, 表示されるか確認する

## 2. whileが回るたびに爆弾を移動, 表示せよ.

- 横方向速度:  $vx = +5$  / 縦方向速度:  $vy = +5$  ※ v はvelocity (速度) の意味
- 爆弾Rectのmove\_ip(vx, vy)メソッドで速度に応じて位置を移動させる
- FPSを50に変更した方が, 画面更新が滑らかになる

# 練習問題

※1問ずつステージング, コミットしよう

## 3. 矢印キーでこうかとんを移動できるようにせよ.

- 押下キーと移動量の対応関係を表す辞書を定義する

辞書のキー：押下キー		辞書の値：移動量タプル	
pg.K_UP	← 上矢印	(0, -5)	← 上に5
pg.K_DOWN	← 下矢印	(0, +5)	← 下に5
pg.K_LEFT	← 左矢印	(-5, 0)	← 左に5
pg.K_RIGHT	← 右矢印	(+5, 0)	← 右に5

- get\_pressed関数によりキーの押下状態リストkey\_lstを取得する
- 複数キー入力に対応すべく  
押下キーと移動量辞書から  
合計移動量を求める

```
合計移動量 = [0, 0]
if key_lst[上矢印]: 合計移動量[1] -= 5
if key_lst[下矢印]: 合計移動量[1] += 5
if key_lst[左矢印]: 合計移動量[0] -= 5
if key_lst[右矢印]: 合計移動量[0] += 5
```

- こうかとんRectをmove\_ip(合計移動量)で移動する

# 練習問題

※1問ずつステージング, コミットしよう

4. こうかといと爆弾が画面の外に出ないようにせよ.

- 画面内or画面外の判定をする関数を実装する
  - 引数 : こうかといRect or 爆弾Rect
  - 戻り値 : 横方向・縦方向の真理値タプル ( True : 画面内 / False : 画面外 )
  - Rectオブジェクトのleft, right, top, bottomの値から画面内・外を判断する
- 更新後の座標が画面外になった場合の挙動
  - こうかとい : 更新前の位置に戻す / 爆弾 : 速度の符号を反転する

5. こうかといが爆弾と衝突したらmain関数からreturnするようにせよ.

- 判定にはRectクラスのcollidirect()を使用する

# 4限：演習問題



# 演習課題：「逃げろ！こうかとん」の改良

- 以下の機能を追加せよ

1. 飛ぶ方向に従ってこうかとん画像を切り替える
2. 時間とともに爆弾が加速する or 大きくなる
3. 着弾するとこうかとん画像が切り替わる（ゲームオーバー前に）
4. 爆弾がこうかとんに近づくように移動する
5. その他，独自の機能

※どの追加機能を実装したのかわかるように，コミットコメントに機能番号を入れること（採点時に必要）

※実装途中やバグありの状態でコミットしないこと

例：`git commit -m "追加機能 1"`

- コードをGitHubにプッシュせよ

※画像ファイルなど，ゲーム実行に必要なものはすべてプッシュすること

# 直前のコミットコメントの修正方法

- 原則：プッシュにより公開した後は修正しない
- 1つ前（＝直前）のコミットコメントを修正する場合

```
git commit --amend -m "修正後のコメント"
```

または

```
git commit --amend
```

 ←エディタでコメントを修正する

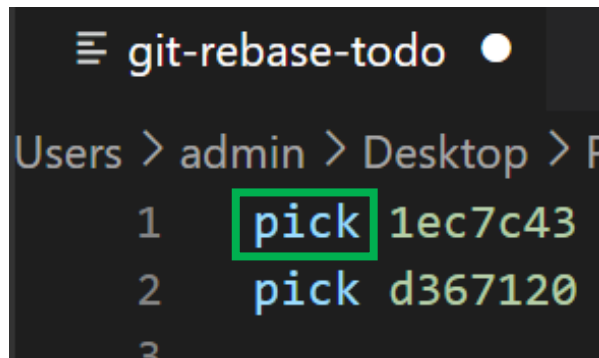
# 直前以外のコミットコメントの修正方法

- 原則：プッシュにより公開した後は修正しない
- 直前以外のコミットコメントを修正する場合

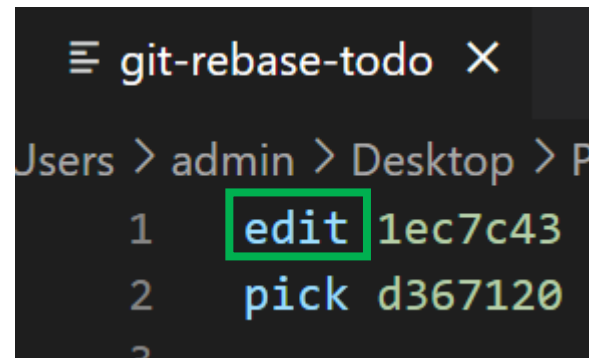
`git rebase -i HEAD~2` ← 2つ前の場合

`hint: Waiting for your editor to close the file...`と表示され、  
裏でエディタが開いているはずなので、してみる

- `pick` を `edit` に書き換え、保存して、閉じる ※2つ前に戻っている状態



```
git-rebase-todo
Users > admin > Desktop > P
1  pick 1ec7c43
2  pick d367120
3
```



```
git-rebase-todo X
Users > admin > Desktop > P
1  edit 1ec7c43
2  pick d367120
3
```

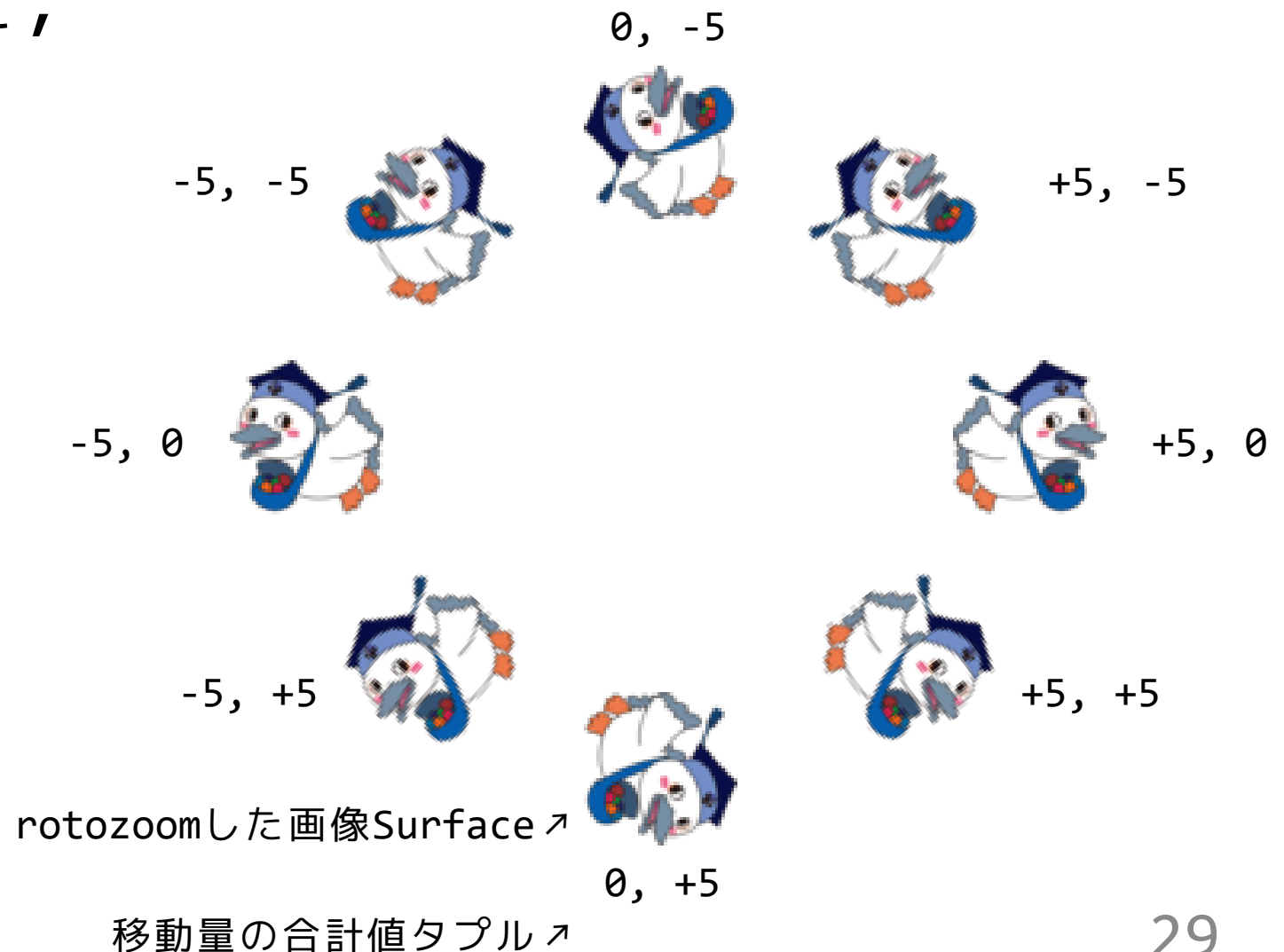
- 前ページを参照して、`amend`でコメントを修正する：`git commit --amend`
- リベースを完了する：`git rebase --continue`

# コーディング時に意識してみよう

- 読みやすさ：空白，空行，文中改行の入れ方
- 簡潔さ↔冗長さ，短さ，一貫性
- 変数名，関数名，クラス名
- 一時変数の利用
- 全体の構造：クラス，関数を定義しているか？
- ループの書き方：`for i in range:` → `for x in lst:`
- 条件文の書き方：`if hoge == True:` → `if hoge:`
- ネストの深さが深すぎないか？
- 必要十分なコメントや型ヒント，docstringがあるか？
- 修正容易性，拡張容易性

# 1. 飛ぶ方向に従ってこうかといん画像を切り替える

- 押下されたキーにしたがって,  
kk\_imgをrotozoomした  
Surfaceをblitする
- 押下キーに対する移動量  
の合計値タプルをキー,  
rotozoomしたSurfaceを  
値とした辞書 →  
を用意しておくとい



## 2. 時間とともに爆弾が加速する or 大きくなる

- 無限に加速or拡大するのはおかしいので, 10段階程度の速度or大きさを用意しておく

- 加速度のリスト:

```
accs = [a for a in range(1, 11)]
```

- 拡大爆弾Surfaceのリスト (一部):

```
for r in range(1, 11):
```

```
    bb_img = pg.Surface((20*r, 20*r))
```

```
    pg.draw.circle(bb_img, (255, 0, 0), (10*r, 10*r), 10*r)
```

```
    bb_imgs.append(bb_img)
```

- tmrの値に応じて, リストから適切な要素を選択する (一部):

```
avx, avy = vx*accs[min(tmr//500, 9)], vy*accs[min(tmr//500, 9)]
```

```
bb_img = bb_imgs[min(tmr//500, 9)]
```

## 4. 爆弾がこうかとんに近づくように移動する

- 爆弾から見て、こうかとんRectがある方向、すなわち移動すべき方向をベクトルとして求める（座標ベクトル間の差）
- 差ベクトルのノルムが $\sqrt{50}$ になるように正規化する
  - 正規化しないと、一瞬でこうかとんに追いついてしまう
  - `move_ip`が小数点以下を無視するため、位置関係によっては爆弾が水平or垂直に移動してしまう（気にしなくてもいい）
- すぐにゲームオーバーにならないように、爆弾とこうかとんの距離（＝正規化前の差ベクトルのノルム）が500未満だったら、慣性として前の方向に移動させる

# 5限：コードレビュー



# コードレビューの手順

※実装途中やバグありの場合は  
`git checkout コミットID`でバグなしの状態まで遡ってみよう  
(Advancedな内容なので、TASA教員に聞いてみよう)

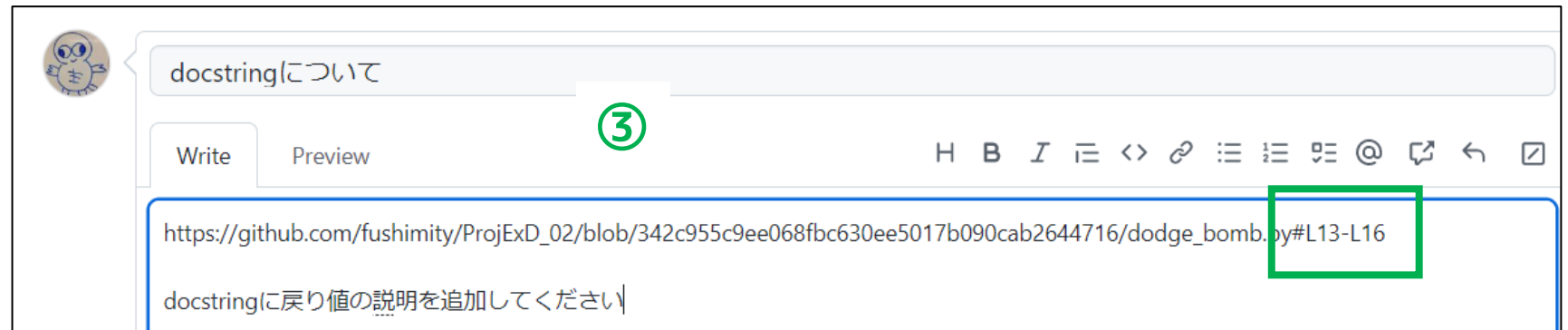
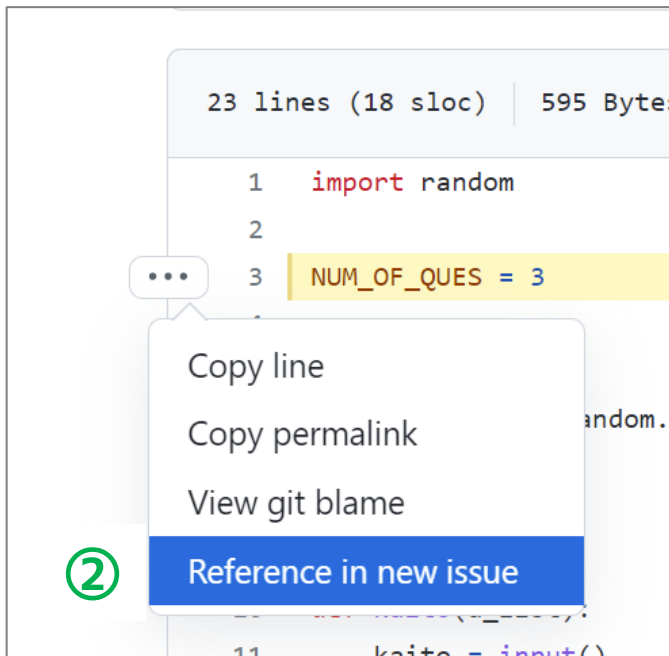
1. グループメンバー1人の公開リポジトリURLを入手する
  - 各自、最低1人のコードをレビューし、最低1人にレビューされることになる
2. 次ページを参照し、URLをもとにコードをクローンする
3. 次ページを参照し、ローカルでゲームを実行してみる
4. コードを見て、リーダブルコードやコード規約の観点で修正すべき点を挙げる
5. 次々ページを参照し、Issueにより修正すべき点を1つずつ送信する
  - 修正すべき点がない完璧なコードの場合、他のメンバーにレビューを依頼する
  - それでも修正すべき点がない場合、TASA、教員にレビューを依頼する
6. 自分の作業ディレクトリに戻る
7. Issueを受信したら、必要に応じてコードを修正しプッシュする

# クローンと実行

- git bashでProjExD2023フォルダに移動する：`cd ../`
- 他メンバーのリポジトリをクローンする：  
`git clone https://github.com/メンバーのアカウント名/ProjExD_02.git`  
※ex02フォルダではなく、ProjExD2023フォルダでcloneすること
- クローンしたコードをVScodeで開き、「Ctrl+F5」で実行する
- 用が済んだら、自分のリポジトリのフォルダに戻る：`cd ../ex02`

# Issueを送る手順

1. グループメンバーのGitHubのページにアクセスし、当該コードを開く  
[https://github.com/fushimity/ProjExD\\_02/blob/main/dodge\\_bomb.py](https://github.com/fushimity/ProjExD_02/blob/main/dodge_bomb.py)
2. コードの該当行を選択し、「・・・」→「Reference in new issue」をクリックする
3. タイトルとコメントを書く
  - 対象が複数行の場合、手動で行数を追加する：「#L13」→「#L13-L16」



# Issueを送る手順

4. Previewで確認する

5. 「Submit new issue」をクリックし送信する



docstringについて

Write Preview ④

ProjExD\_02/dodge\_bomb.py  
Lines 13 to 16 in 342c955

```
13      """
14      オブジェクトが画面内か画面外かを判定し、真理値タプルを返す
15      引数1 area: 画面SurfaceのRect
16      引数2 obj: オブジェクト（爆弾，こうかとん）SurfaceのRect
```

docstringに戻り値の説明を追加してください

⑤

Styling with Markdown is supported

Submit new issue

# Issueが届く

- Issuesタブから，Issueコメントを読み，対応する



Issues 1 Pull requests Actions Projects Security Insights Settings

## docstringについて #1 ← Issue番号

Open fushimity opened this issue now · 0 comments

fushimity commented now

ProjExD\_02/dodge\_bomb.py  
Lines 13 to 16 in 342c955

```
13      """"
14      オブジェクトが画面内か画面外かを判定し，真理値タプルを返す
15      引数1 area: 画面SurfaceのRect
16      引数2 obj: オブジェクト（爆弾，こうかとん）SurfaceのRect
```

docstringに戻り値の説明を追加してください

Write Preview

Leave a comment

# コード修正後の手順

- 自分の作業ディレクトリ ex02 でコード修正する

1. 修正が終わったらステージングする：`git add ファイル名`

2. ステージングされた内容をコミットする

※重要：コミットコメントに、対応したIssue番号を付けること

`git commit -m "コメント #Issue番号"`

例：`git commit -m "変数名統一 #2"`

- ・「#」の前に半角スペースが必要
- ・「#Issue番号」は半角で入力する

これにより、Issueと対応コミットがGitHub上で紐づけられる

3. リモートリポジトリにプッシュする：`git push origin main`

# チェック項目

1. 追加機能の実装数(1機能2点 / 4機能まで) [0 ~ 8]
2. リーダブルコード [0 ~ 4]
  - 空白, 空行が適切か
  - 必要十分なコメント
  - 関数の実装(`check_bound`以外)
  - 関数アノテーション(引数, 戻り値の型ヒント), `docstring`の有無
3. Issue [0 ~ 3]
  - 受信
  - コード修正
  - Issue番号を付してコミット
4. 提出物不備は1点ずつ減点

# 提出物

学籍番号は, 半角・大文字で

- ファイル名 : C0A22XXX\_kadai02.pdf
- 内容 : 以下の順番でPDFを結合して提出すること
  - コミット履歴  
[https://github.com/fushimity/ProjExD\\_02/commits/main](https://github.com/fushimity/ProjExD_02/commits/main)
  - コードの最終版  
[https://github.com/fushimity/ProjExD\\_02/blob/main/dodge\\_bomb.py](https://github.com/fushimity/ProjExD_02/blob/main/dodge_bomb.py)
  - Issue一覧  
[https://github.com/fushimity/ProjExD\\_02/issues](https://github.com/fushimity/ProjExD_02/issues)

自分のアカウント名



# 提出物の作り方

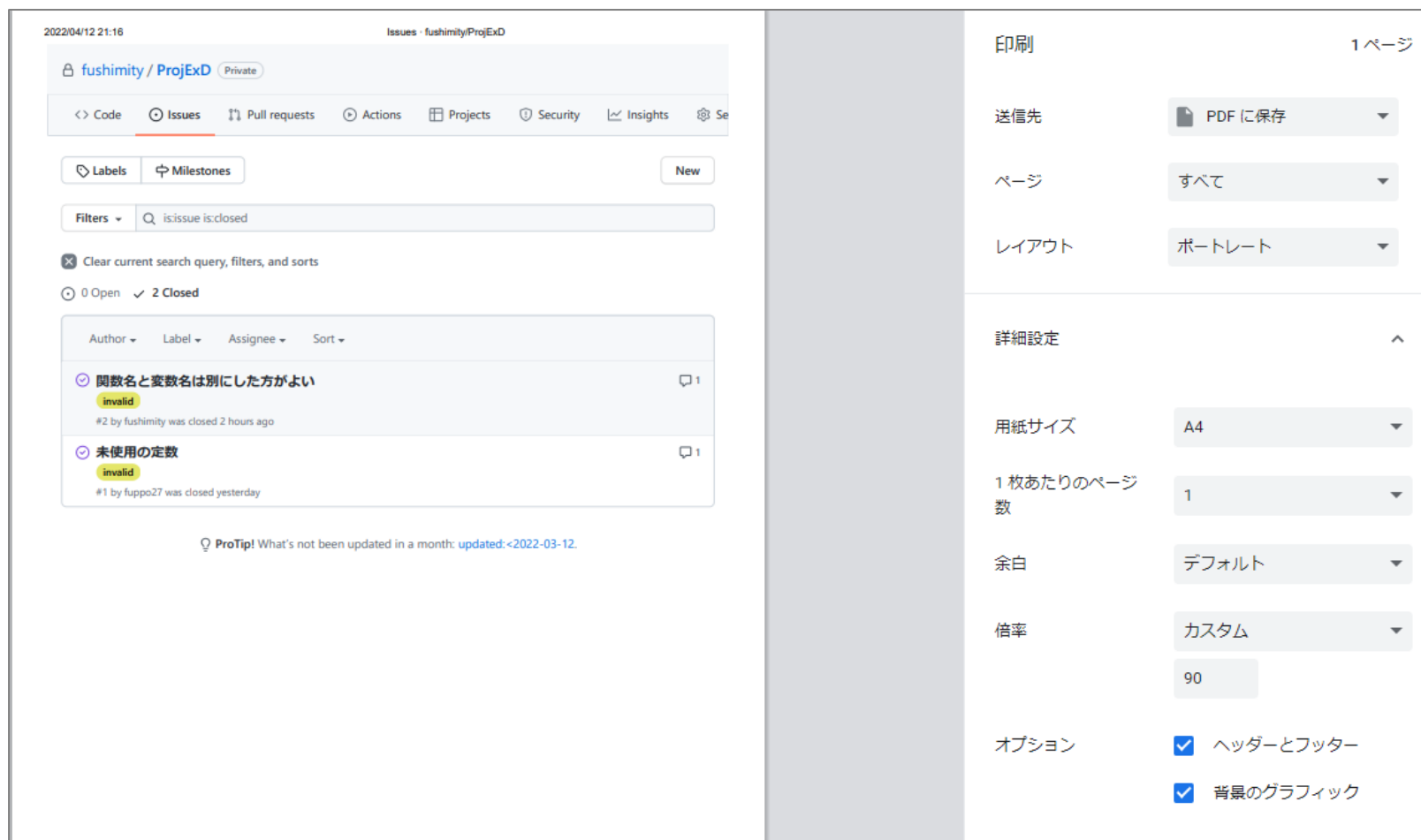
※スクショは認めません。

以下の手順に従ってPDFを作成し、提出すること

1. ChromeでPDFとして保存する（次ページを参照）
2. 以下のURLから各PDFをマージする  
[https://www.ilovepdf.com/ja/merge\\_pdf](https://www.ilovepdf.com/ja/merge_pdf)
3. ファイル名を「C0A22XXX\_kadai02.pdf」として保存する

# ChromeでPDFとして保存する方法

1. 該当ページを表示させた状態で「Ctrl+P」
2. 以下のように設定し、「保存」をクリックする



←送信先：PDFに保存

←ページ：すべて

←レイアウト：ポートレート

←用紙サイズ：A4

←余白：デフォルト

←倍率：90

←両方チェック

# チェックの手順

※基本的に再提出できません。どうしてもの場合は要相談。

1. 受講生：提出物（pdf）を作成し，Moodleに提出する
  2. 受講生：担当TASAに成果物（ゲーム）を見せに行く
  3. TASA：提出物とゲームのデモを確認し，点数を確定する
  4. 受講生：帰る
  5. FSM：近日中に課題と点数を確認し，Moodleに登録する
- 時間内にチェックが終わらなそうな場合は，提出物をMoodleに提出し帰る  
（次回までor次回の3限にチェックされる）

← 時間外提出扱いになり  
割引いて採点するので  
できるだけチェックを  
受けること