

# 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

- ex2.zipをProjExDフォルダにDLし, 展開する

## 【配布物配置後のディレクトリ構造】

- ProjExD/

- sample.py

- ex1/

- ex2/

- dodge\_bomb.py . . . 逃げろ！こうかтон

- fig/

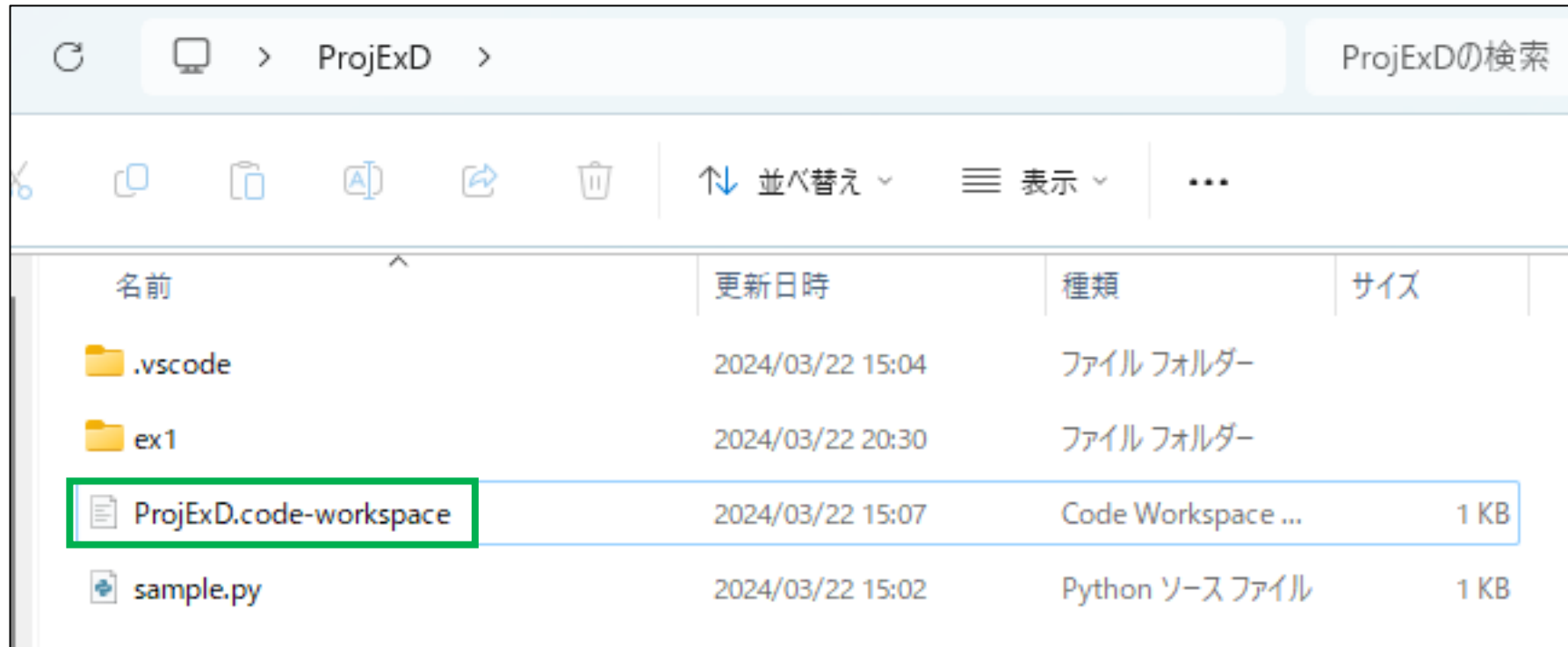
- pg\_bg.jpg . . . 背景画像

- {0, ..., 9}.png . . . こうかтон画像

本日の配布物

# VScodeの起動

- ProjExDフォルダ内の「ProjExD.code-workspace」をダブルクリックし、ワークスペースを開く



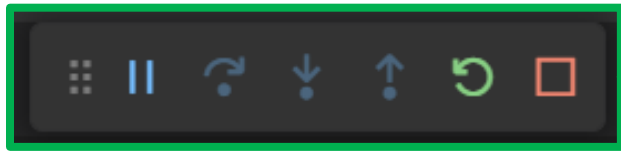
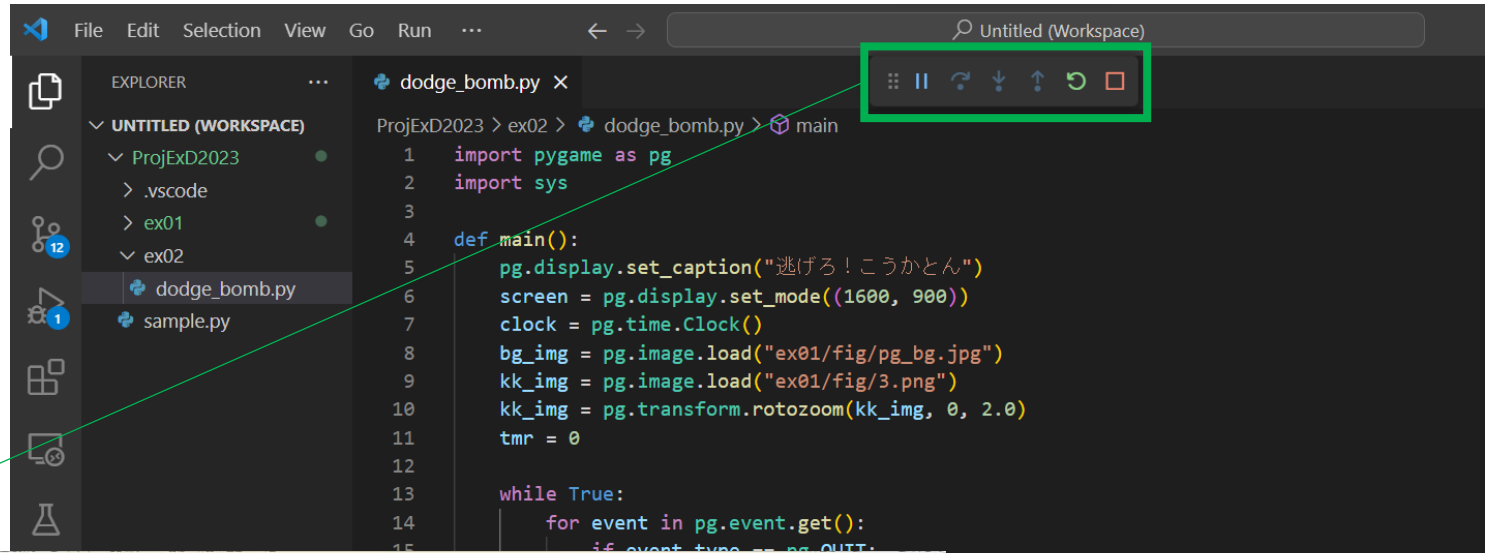
# 前回の復習：git


- 「ProjExD¥ex2」フォルダを右クリックし, Git Bashを起動する
- 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\_2」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_2.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`

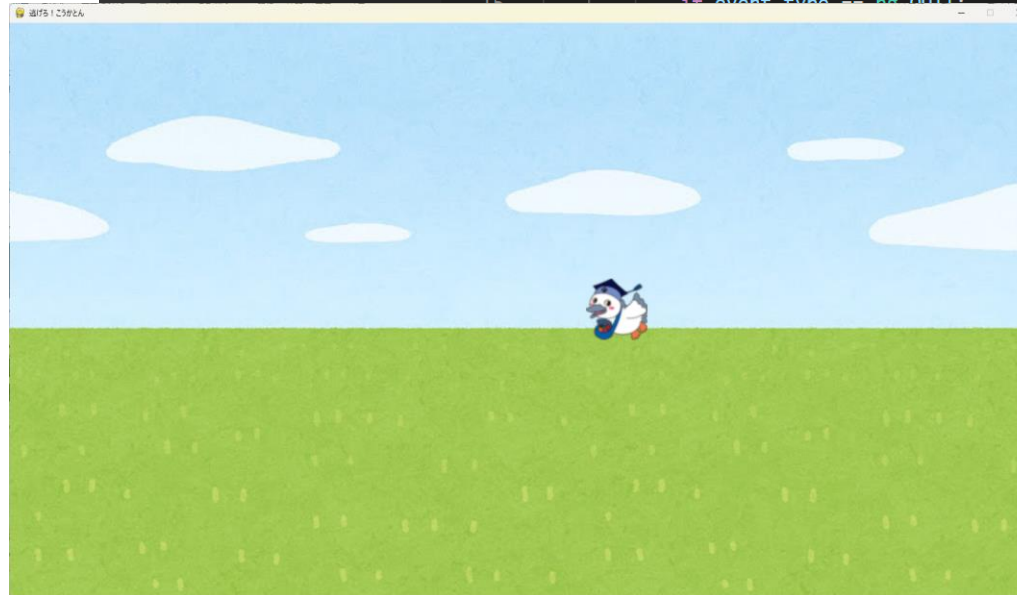
Git for Windowsでない人は  
SSH版をコピー

# 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



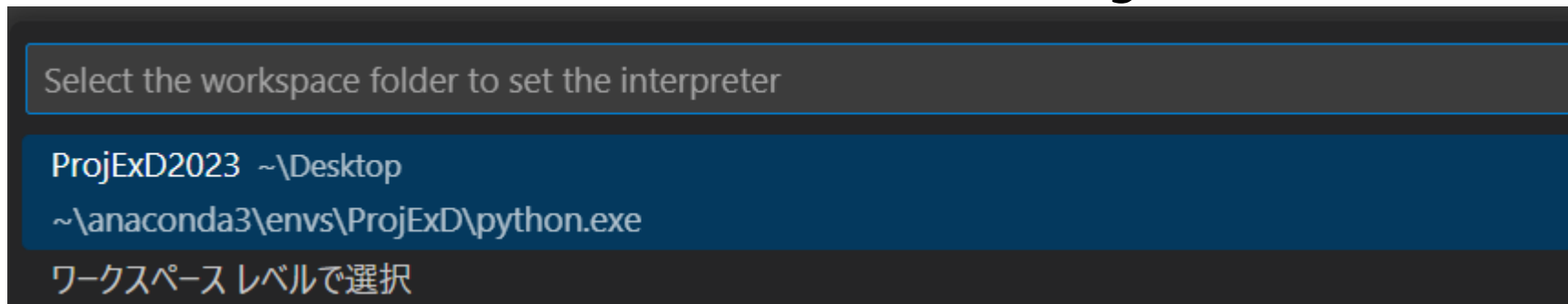
- 「」で終了する



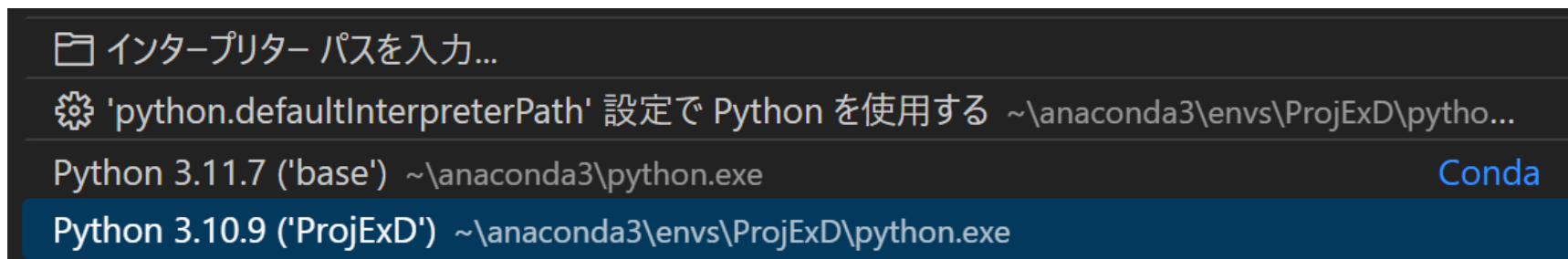


# 実行できなかったら → VScode右下を確認

- 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として「ProjExD」を選ぶ



- インタープリターパスとして「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
- こうかとんSurfaceに対応するRectを取得し，初期座標900，400を設定する

## 【ゲームのループ】

- 背景SurfaceをスクリーンSurfaceに貼り付ける
- 押下キーを取得し，キーに応じて合計移動量を計算する
- こうかとんSurfaceをRectの設定に従い，スクリーンSurfaceに貼り付ける
- 画面を更新する

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(obj_rct: pg.Rect) -> tuple[bool, bool]:
```

```
"""
```

```
引数：こうかとんRectかばくだんRect
```

```
戻り値：タプル（横方向判定結果，縦方向判定結果）
```

```
画面内ならTrue，画面外ならFalse
```

```
"""
```

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

```
if obj_rct.left < 0 or WIDTH < obj_rct.right: # 横方向判定
```

```
    yoko = False
```

```
if obj_rct.top < 0 or HEIGHT < obj_rct.bottom: # 縦方向判定
```


```
    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

check

# 練習  check\_bound

```
def check_bound(obj_rct: Rect) -> tuple ×
```

引数：こうかとんRectかばくだんRect 戻り値：タプル  
（横方向判定結果，縦方向判定結果） 画面内ならTrue，画面外ならFalse

# 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）

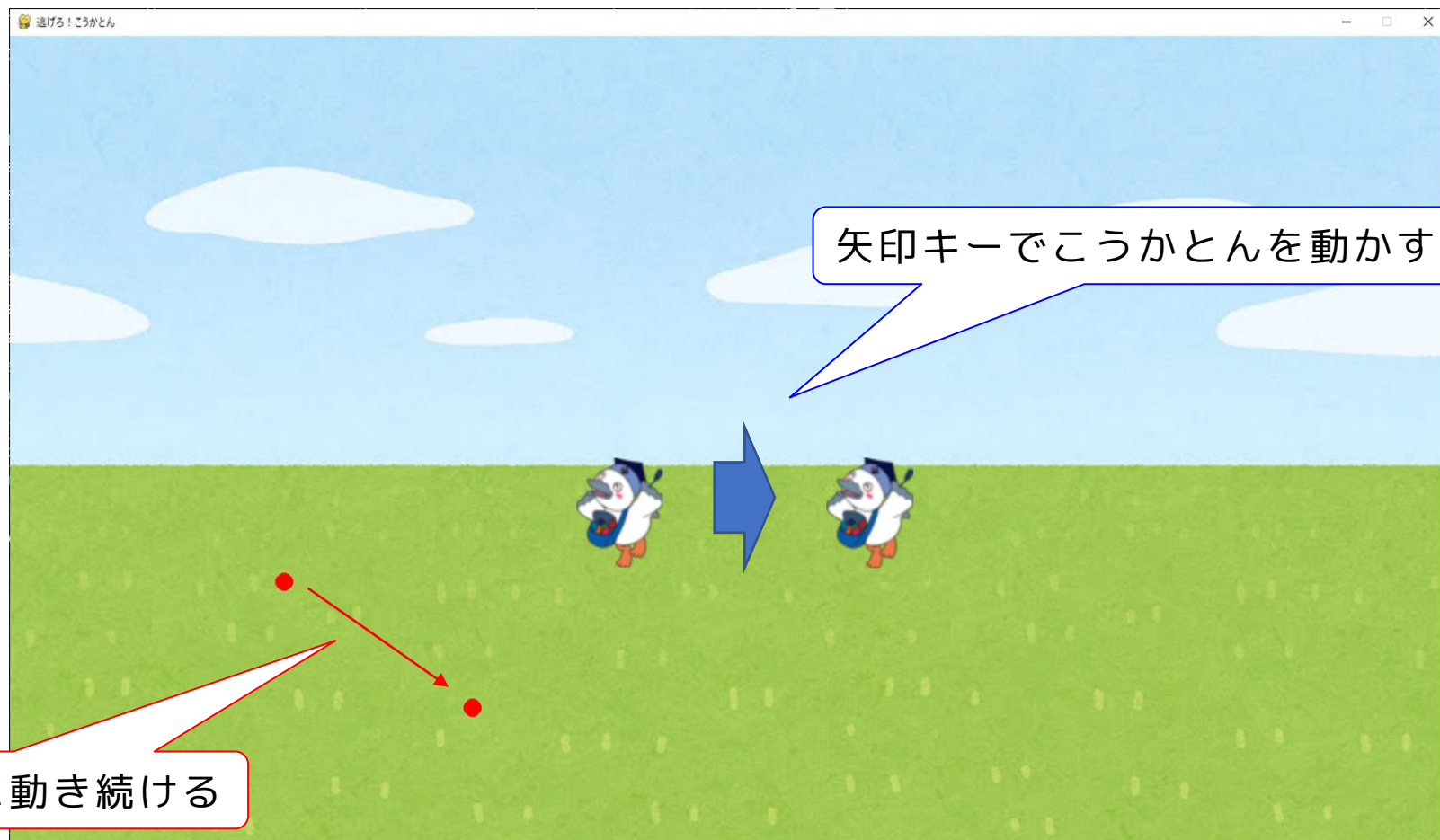
- 空白
  - 余分な空白は入れない
  - 統一する（カンマの後ろ，演算子の前後）
  - 「＝」の位置を合わせるために空白を入れない
  - キーワード引数，デフォルト引数の「＝」前後は空白を入れない
- 命名規則
  - 意味を表した単語にする（よく知られた省略は可能）
  - クラス名：パスカルケース（例：`SampleClass`）
  - 関数名，変数名：スネークケース（例：`sample_func`）
  - 定数名：全て大文字でアンダースコアでつなげる（例：`SAMPLE_CONST`）
  - 予約語などと被る場合：後ろにアンダースコアを付ける（例：`id_`）
- 条件文
  - 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. 押下キーに応じてこうかとんを動かす部分のコードにおいて, 移動量辞書を定義することで, 冗長な記述を改めよ
  - 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

# 練習問題

※1問ずつステージング, コミットしよう

2. 爆弾Surfaceを作成し, ランダムな位置に配置し,  
whileが回るたびに爆弾を移動, 表示せよ

- 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
- 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

# 練習問題

※1問ずつステージング, コミットしよう

3. こうかたとん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 : こうかたん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
  - こうかたん : 更新前の位置に戻す
  - 爆弾 : 横 ( 縦 ) 方向に出そうになったらvx ( vy ) の符号を反転する

4. こうかたんと爆弾と衝突したら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

- 以下の追加機能を実装せよ

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拡大，加速する
3. ゲームオーバー画面（こうかとん画像の切替と「GameOver」表示）
4. 追従型爆弾（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

- 注意事項

- 関数を定義して実装すること（何でも関数にすれば良いわけではないが，練習ということで）
- 型ヒントやdocstringを導入すると，点数が高くなる
- 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）
- 頻繁にコミットする場合は，完成なのか途中なのかわかるようにすること

例：`git commit -m "追加機能1 途中"`

# 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

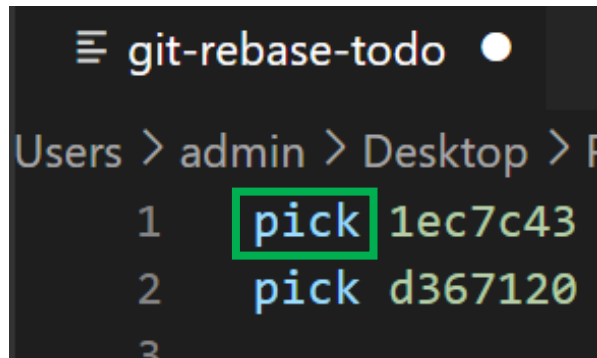
# 直前以外のコミットコメントの修正方法

- 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

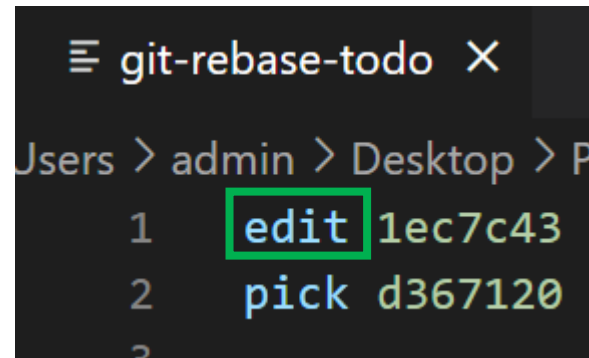
`git rebase -i HEAD~2` ← 2つ前の場合

`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
```

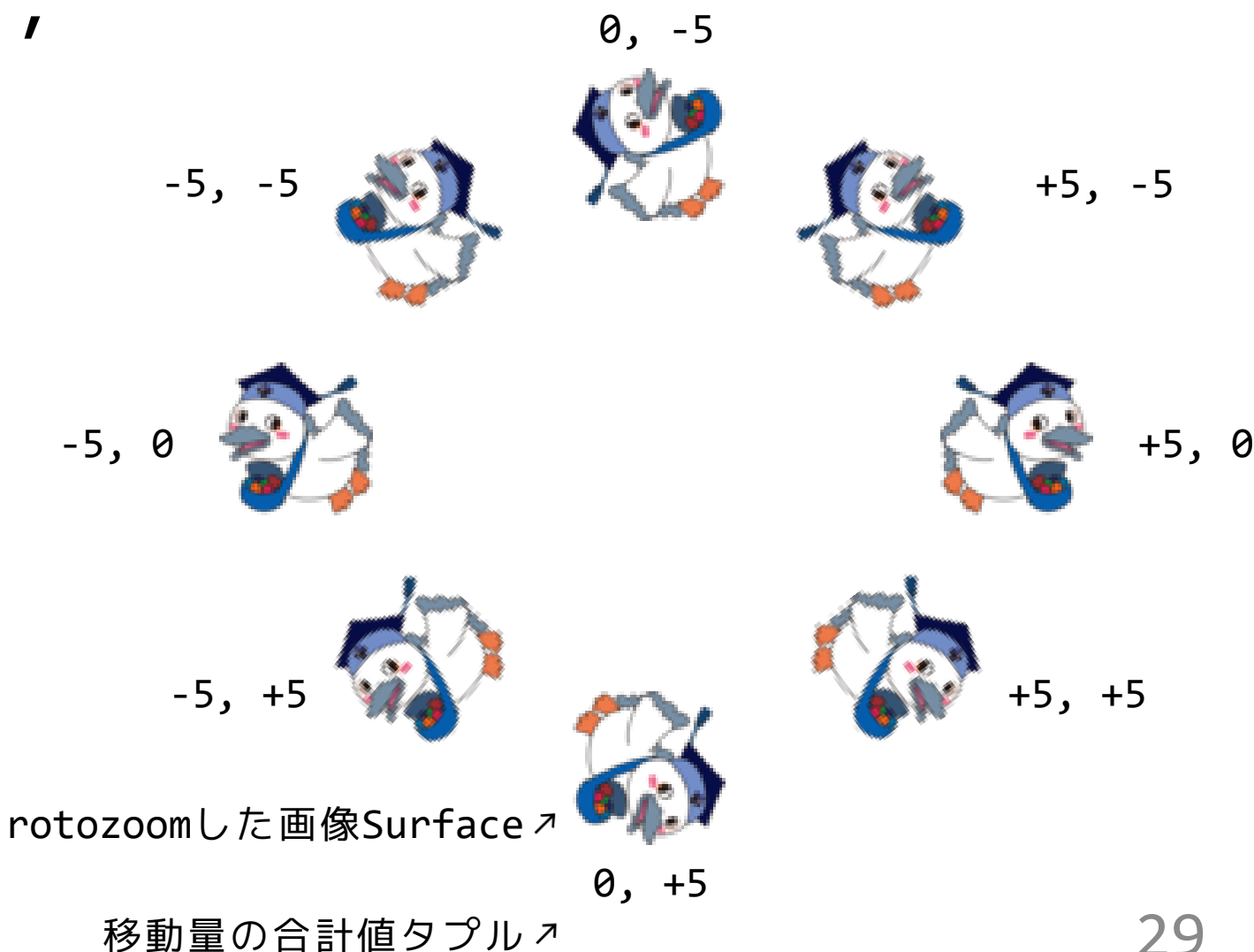
- 前ページを参照して, amendでコメントを修正する : `git commit --amend`
- リベースを完了する : `git rebase --continue`

# 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  
値とした辞書を準備する
- 辞書を返す関数を実装する



## 2. 時間とともに爆弾が拡大，加速する

- 無限に拡大，加速するのはおかしいので，10段階程度の大きさ，加速度を準備する

- 加速度のリスト：

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

- 拡大爆弾Surfaceのリスト（一部掲載）：

```
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)
```

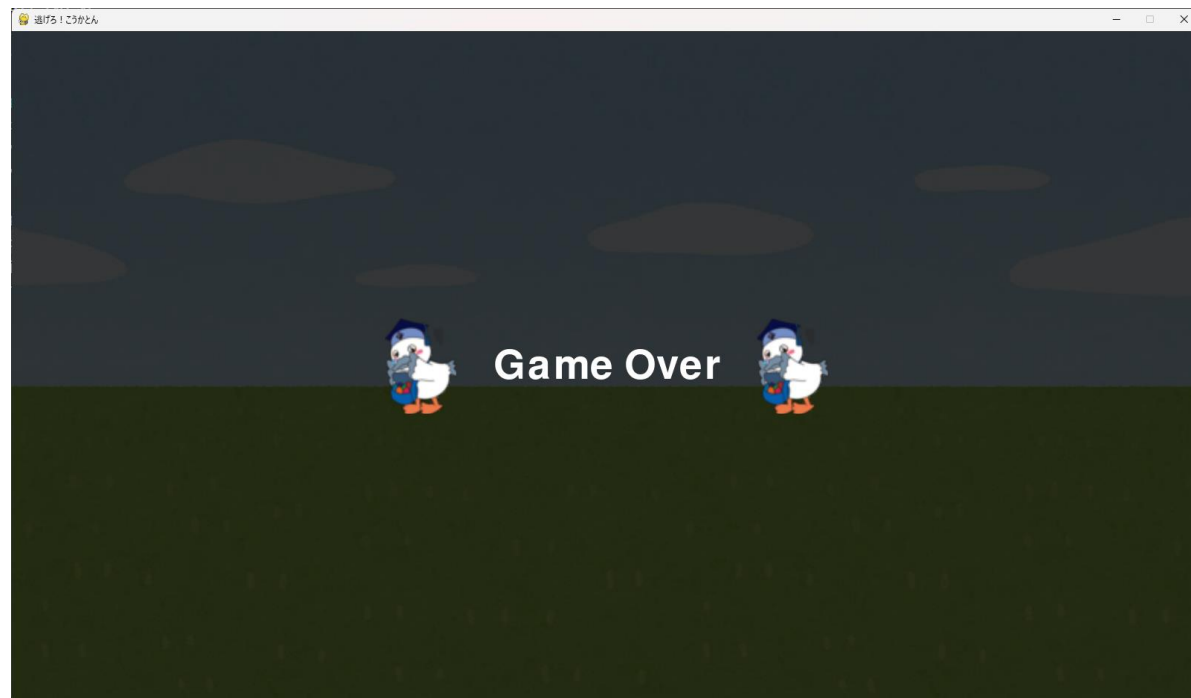
- これらのリストのタプルを返す関数を実装する
  - tmrの値に応じて，リストから適切な要素を選択する（一部掲載）：

```
avx = vx*bb_accs[min(tmr//500, 9)]
```

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

### 3. ゲームオーバー画面（こうかどん画像の切替と「GameOver」表示）

- こうかどんに爆弾が着弾した際に、
  - 画面をブラックアウトし、
  - 泣いているこうかどん画像と
  - 「Game Over」の文字列を
  - 5秒間表示させる関数を実装する
- ブラックアウトには
  - `draw.rect`関数で四角を描画する
    - ↑ 前回演習資料P.34を参考にする
  - `Surface.set_alpha(透明度)`メソッドで半透明にする
- 「Game Over」の文字列表示には
  - ↑ 前回演習資料P.33を参考にする



↑  
あくまでも例です

## 4. 追従型爆弾 (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300未満だったら, 慣性として前の方向に移動させる
- 上記の条件を満たす爆弾の速度ベクトル (タプル) を返す関数を実装する

速度ベクトルのノルム:  $\sqrt{(5^2+5^2)}$



# 5限：コードレビュー

4限演習課題のコミットをGitHubにプッシュせよ

※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)

# コードレビューの手順

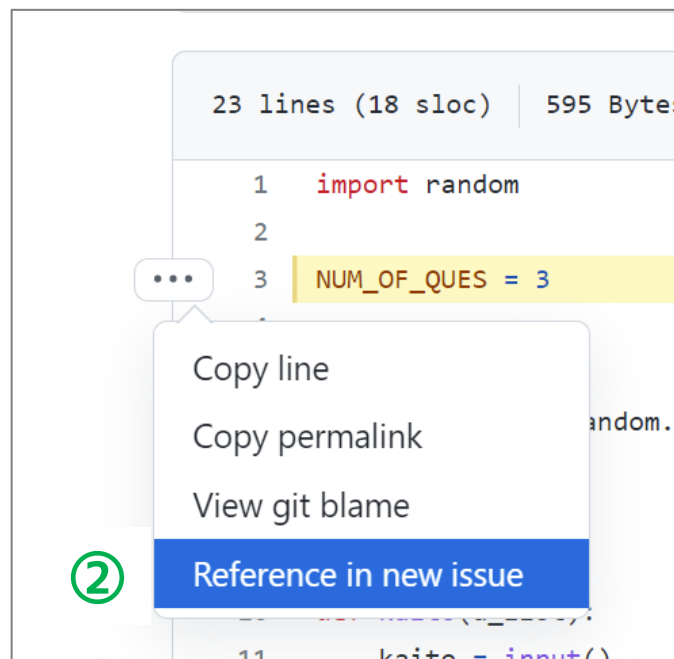
1. グループメンバー1人の公開リポジトリURLを入手する
  - 各自、最低1人のコードをレビューし、最低1人にレビューされることになる
2. 次ページを参照し、URLをもとにコードをクローンする
3. 次ページを参照し、ローカルでゲームを実行してみる
4. コードを見て、リーダブルコードやコード規約の観点で修正すべき点を挙げる
5. 次々ページを参照し、Issueにより修正すべき点を1つずつ送信する
  - 修正すべき点がない完璧なコードの場合、他のメンバーにレビューを依頼する
  - それでも修正すべき点がない場合、TASA、教員にレビューを依頼する
6. 自分の作業ディレクトリに戻る
7. 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ProjExD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_2.git`  
※ex2フォルダではなく、ProjExD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 ex2`

# 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\\_2/blob/main/dodge\\_bomb.py](https://github.com/fushimity/ProjExD_2/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



The screenshot shows the GitHub issue submission interface. At the top, there is a text input field containing "docstringについて". Below the input field are two tabs: "Write" and "Preview", with the "Preview" tab selected and marked with a green circle containing the number 4. The preview area displays the file path "ProjExD\_02/dodge\_bomb.py" and "Lines 13 to 16 in 342c955". The code snippet shows a docstring for a function that checks if an object is on the screen. Below the code, there is a text input field with the placeholder text "docstringに戻り値の説明を追加してください". At the bottom right, there is a green button labeled "Submit new issue", which is marked with a green circle containing the number 5. At the bottom left, there is a small icon and the text "Styling with Markdown is supported".

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

Submit new issue

Styling with Markdown is supported

# Issueが届く

- Issuesタブから，Issueコメントを読み，対応する



The screenshot shows the GitHub interface for an issue titled "docstringについて" (About docstring). The issue is marked as "Open" and was opened by "fushimity". A green box highlights the issue number "#1", with a green arrow pointing to the text "Issue番号" (Issue number). Below the issue title, a comment from "fushimity" is shown. The comment includes a code snippet from a file named "ProjExD\_02/dodge\_bomb.py", lines 13 to 16. The code is a docstring for a function that determines if an object is on-screen or off-screen. The comment text says "docstringに戻り値の説明を追加してください" (Please add the return value explanation back to the docstring).

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 ex2 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 #番号"`

- ・「#」の前に半角スペースが必要
- ・「#番号」は半角で入力する

例：`git commit -m "変数名統一 #1"`

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（3点満点x最大4機能）
  - 追加機能が正常に動いている：+1点
  - 関数を用いて実装している：+1点
  - 追加実装した関数に型ヒント, docstringがある：+1点
2. 全体がリーダブルコードか？
  - 空白, 空行が適切か：+1点
  - 必要十分なコメント：+1点
3. Issue番号を付してコミット：+1点
4. 提出物不備（ファイル名, クリックابل, 内容物）は1点ずつ減点



# 提出物

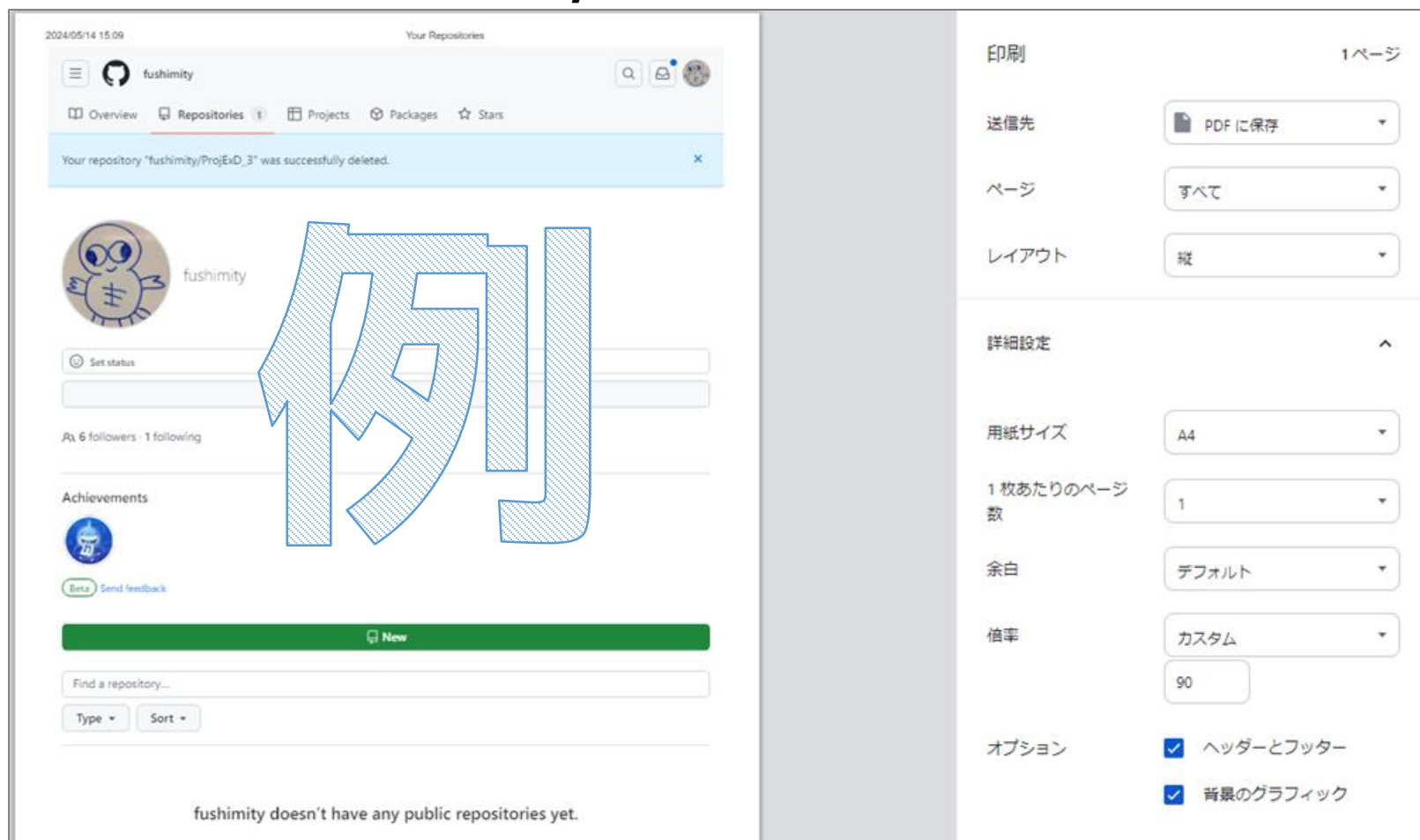
学籍番号は, 半角・大文字で

- ファイル名 : C0A23XXX\_kadai2.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\\_2/commits/main](https://github.com/fushimity/ProjExD_2/commits/main)
  - コードの最終版  
[https://github.com/fushimity/ProjExD\\_2/blob/main/dodge\\_bomb.py](https://github.com/fushimity/ProjExD_2/blob/main/dodge_bomb.py)
  - Issue一覧  
[https://github.com/fushimity/ProjExD\\_2/issues](https://github.com/fushimity/ProjExD_2/issues)

自分のアカウント名

# 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：縦

←用紙サイズ：A4

←余白：デフォルト

←倍率：90

←両方チェック

# 提出物の作り方

※スクショ画像は認めません。  
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「C0A23XXX\_kadai2.pdf」として保存する

# チェックの手順

※基本的に、チェック後の修正・再提出できませんので、慎重に提出物PDFを作ること。どうしても場合は要相談。

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