

# プロジェクト演習 テーマD

## 第4回

担当:CS学部 講師 伏見卓恭

連絡先:fushimity@edu.teu.ac.jp

# 授業の流れ

- 第 1回:実験環境の構築／Python, Gitの復習／CUIゲームの開発
- 第 2回:Tkinterの基礎
- 第 3回:TkinterによるGUIゲーム開発
- 第 4回:PyGameの基礎
- 第 5回:PyGameによるゲーム開発
- 第 6回:ゲーム開発演習
- 第 7回:成果発表

# 本日のお品書き

1. 実験環境の構築(Pygame)
2. Pygameの基礎
3. Pygameを使ってゲームの開発(前半)

# 3限：環境構築

# pygameモジュールのインストール

- Anaconda prompt, または, コマンドプロンプトで, 以下を入力する

```
(ProjExD) C:\Users\fsmtkys>pip install --user pygame
```

- インストールされたものを確認する

```
(ProjExD) C:\Users\fsmtkys>pip list
```

Package	Version
certifi	2021.10.8
pip	21.2.4
pygame	2.1.2
setuptools	58.0.4
wheel	0.37.1
wincertstore	0.2

PyGameドキュメント(日本語訳):

<http://westplain.sakuraweb.com/translate/pygame/>

# 3限: Pygameの基礎

mainブランチになっていることを確認する

ProjExD2022/ex04/フォルダを作成する

# 初期化と終了

- pygameパッケージ(モジュール)をimportする

```
import pygame as pg
```

- pygameモジュールを初期化する

```
pg.init()
```

- pygameモジュールの初期化を解除する

```
pg.quit()
```

その後、プログラムを「`sys.exit()`」で終了する

```
1  import pygame as pg
2  import sys
3
4
5  > def main():...
48
49  if __name__ == "__main__":
50      pg.init()          # モジュールを初期化する
51      main()
52      pg.quit()          # モジュールの初期化を解除する
53      sys.exit()         # プログラムを終了する
```

# 画面, スクリーン

- displayモジュールの関数
  - set\_caption(タイトル文字列): ウィンドウのタイトルを設定する
  - set\_mode(幅と高さのタプル): 画面用のSurfaceクラスのオブジェクトを生成する
  - update(): 画面を更新する

```
pg.display.set_caption("初めてのPyGame") # タイトルバーに「初めての. . .」を表示する  
scrn_sfc = pg.display.set_mode((800,600)) # 800x600の画面Surfaceを生成する
```



# Surfaceクラス

- PyGameでは、読み込んだ画像、作成した図形、文字などはSurfaceクラスのインスタンスとして表現され、別のSurfaceクラスに貼り付ける(blitする)ことができる
- おおもとのスクリーンsurfaceにインスタンスを貼り付けることで、表示させることができる.

- インスタンスメソッド

- blit(別のsurface, 位置座標rect): surfaceに別のsurfaceを貼り付ける
- get\_rect(): surfaceが存在する範囲をRectクラスのオブジェクトとして返す
- fill(色を表すタプル): surfaceを一色に塗りつぶす

# Rectクラス

- Pygameでは, Rectクラスのオブジェクトを使用してsurface(画面, 画像, 図形, 文字など)を描画する矩形範囲を設定, 変更する
  - インスタンス変数
    - 位置に関するもの: top, left, bottom, right, center, centerx, centery
    - 大きさに関するもの: size, width, height
  - インスタンスメソッド
    - move\_ip(vx, vy): 指定した距離分移動させる
    - clamp(別のrectオブジェクト): 別のrectオブジェクトの枠内に移動させたrectオブジェクトを新規に作成する
    - colliderect(別のrectオブジェクト): 別のrectオブジェクトと重なっている場合trueを返す

※SurfaceオブジェクトとRectオブジェクトは対(ペア)で用いることが多い

# 画像

- imageモジュールの関数

- load(ファイルパス): 指定したパスのファイルを読み込み, 画像を描画したSurfaceクラスのオブジェクトを生成する

- transformモジュールの関数

- scale(画像surface, (幅, 高さ)): 拡大縮小
- rotate(画像surface, 回転角): 回転
- rotozoom(画像surface, 回転角, 大きさの比率): 回転 + 拡大縮小

```
tori_sfc = pg.image.load("fig/6.png") # Surface
tori_sfc = pg.transform.rotozoom(tori_sfc, 0, 2.0)
tori_rct = tori_sfc.get_rect() # Rect
tori_rct.center = 700, 400
scrn_sfc.blit(tori_sfc, tori_rct)
```

# 文字

- Fontクラス: 文字列のフォントを規定するクラス
  - Font(フォント名, サイズ): フォントオブジェクトを生成する
  - render(文字列, True, 色タプル): 指定した色の文字列を書いたSurfaceクラスのオブジェクトを生成する

※Pygameでは作成済みのsurfaceに直接文字を描画することはできない

## • 文字描画の手順

1. フォント, サイズを指定する: Font()

```
fonto = pg.font.Font(None, 80) # フォント設定用のオブジェクト
```

2. 文字を書いたSurfaceを生成する: render()

```
txt = fonto.render(str(tmr), True, WHITE) # 白色で文字列を書いたSurfaceを生成する
```

3. 描画面surfaceに, 2の文字surfaceを貼り付ける: blit()

```
scrn_sfc.blit(txt, (300, 200)) # 文字列Surfaceを画面Surfaceに貼り付ける
```

# 時間

- Clockクラス: 時間管理, 計測用のクラス

- Clock(): (コンストラクタ) 時間計測用オブジェクトを生成する
- tick(フレームレート): 指定したフレーム秒の遅延を発生させる

```
clock = pg.time.Clock()           # 時間計測用のオブジェクト
```

```
clock.tick(1)                     # 1fpsの時を刻む
```

フレームレート(frame per second):

動画において, 単位時間(1秒)あたりに処理させるフレームすなわち「コマ」の数を示す頻度の数値のこと

# イベント

- eventモジュールの関数

- get(): イベントキューから全てのイベント情報を取得する
  - ※ 取得されたイベントはイベントキューから削除される

```
for event in pg.event.get(): # イベントを繰り返して処理
    if event.type == pg.QUIT: return # ウィンドウの×ボタンをクリックしたら
```

- event.type: イベントの種類

- QUIT: ×ボタンのクリック
- KEYDOWN: キーの押下

- event.key: キーの種類

```
if event.type == pg.KEYDOWN and event.key == pg.K_F1:
    scrn_sfc = pg.display.set_mode((800, 600), pg.FULLSCREEN)
if event.type == pg.KEYDOWN and event.key == pg.K_ESCAPE:
    scrn_sfc = pg.display.set_mode((800, 600))
```

# キーボード, マウス

- keyモジュールの関数

- `get_pressed()`: すべてのキーの入力状態に関する辞書を返す
  - 辞書のキー: キーボード定数 (例: `K_UP`, `K_SPACE`, `K_a`, `K_0`, `K_LSHIFT`, `K_F1`)
  - 辞書の値: `True` of `False`

```
key_lst = pg.key.get_pressed()
print(key_lst[pg.K_SPACE])
```

- mouseモジュールの関数

- `get_pos()`: マウスカーソルの位置を返す

```
print(pg.mouse.get_pos())
```

# 図形

- drawモジュールの関数

- rect(描画用surface, 色, 四角形の範囲): 四角形を生成する
- polygon(描画用surface, 色, 点リスト): 多角形を生成する
- circle(描画用surface, 色, 中心座標, 半径): 円を生成する
- line(描画用surface, 色, 始点座標, 終点座標): 直線の線分を生成する



# blitによるsurface合成の例

```
draw_sfc = pg.Surface((100,100))
```

←描画用surface(幅:100, 高さ:100)を生成する

```
pg.draw.circle(draw_sfc, (255, 0, 0), (50,50), 50)
```

```
pg.draw.circle(draw_sfc, (191, 0, 0), (50,50), 40)
```

```
pg.draw.circle(draw_sfc, (127, 0, 0), (50,50), 30)
```

```
pg.draw.circle(draw_sfc, ( 63, 0, 0), (50,50), 20)
```

```
pg.draw.circle(draw_sfc, (  0, 0, 0), (50,50), 10)
```

←描画用surfaceに  
赤((255,0,0))で  
位置(横:50, 縦:50)に  
半径50の円を描画する

```
scrn_sfc.blit(draw_sfc, (100,100))
```

```
scrn_sfc.blit(draw_sfc, (200,200))
```

```
scrn_sfc.blit(draw_sfc, (300,300))
```

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

←描画用surfaceに  
黒((0,0,0))で  
位置(横:50, 縦:50)に  
半径10の円を描画する

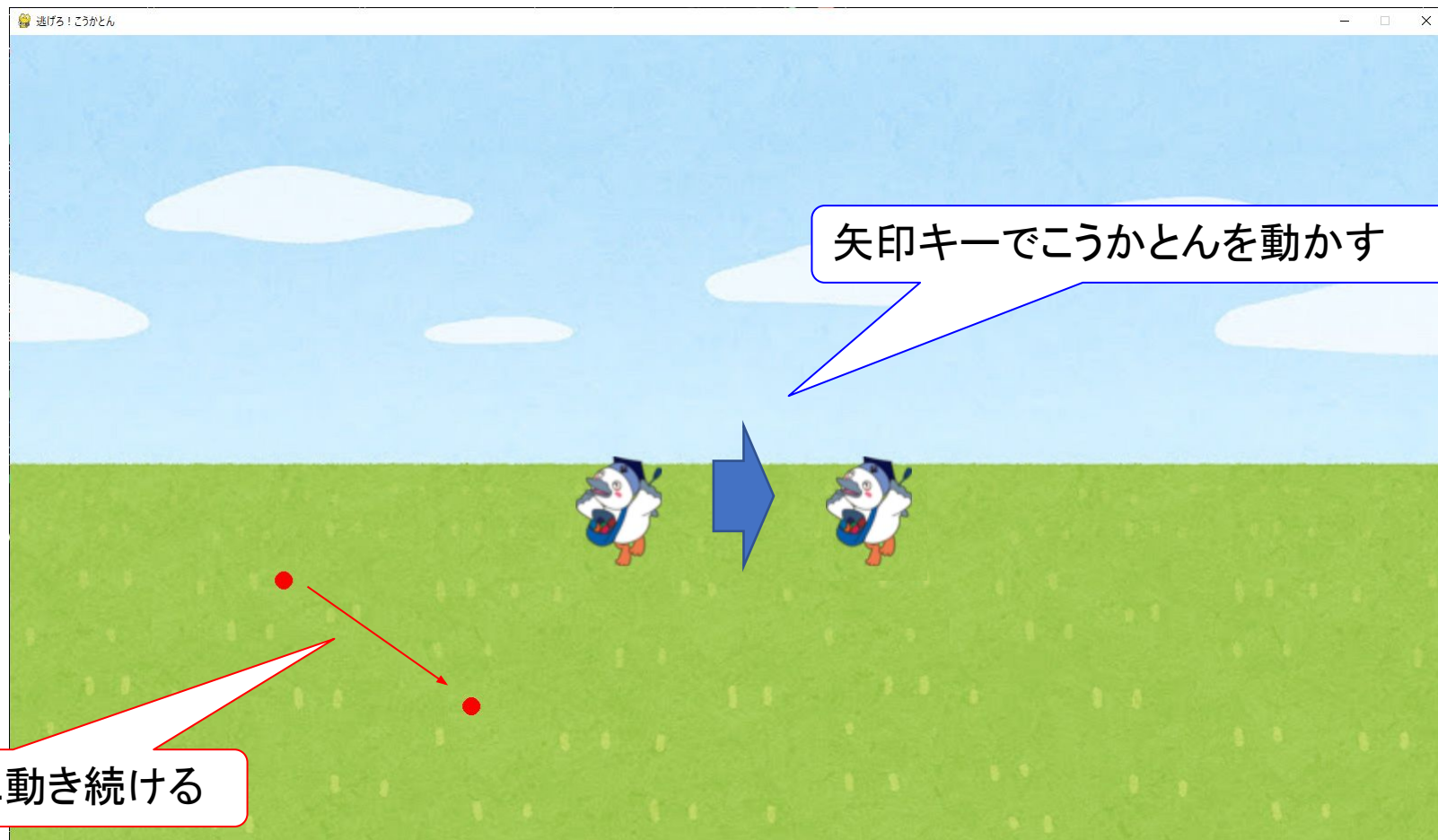
これがdraw\_sfc



↑  
描画用surfaceであるdraw\_sfcを  
画面用surfaceであるscrn\_sfcの  
位置(横:300, 縦:300)に  
描画する

# 練習問題: ex04/dodge\_bomb.py

- 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: ex04/dodge\_bomb.py

1. 「逃げろ! こうかとん」用のウィンドウを生成し,  
幅:1600, 高さ:900, 背景:pg\_bg.jpgのsurfaceを生成する
  - ※ 「clock = pg.time.Clock() 」と「clock.tick(0.5)」を入れると, 2秒間表示され確認できる
  - ※ ゲーム画面がディスプレイからはみ出す場合は, 幅と高さを調整する
2. 無限ループの中で,
  - 背景画像をblitする
  - 「X」ボタンが押されたらmain関数からreturnするような処理を追加する
  - display.update()する
  - 1000fpsの時を刻む
3. figファルダ内の好きなこうかとん画像を横:900, 縦:400の座標に, 2倍に拡大して表示させる
  - 拡大には, transform.rotozoom()を使う

# 練習問題: ex04/dodge\_bomb.py

4. 矢印キーでこうかとんを移動できるようにする
  - Up: 上に1 / Down: 下に1 / Left: 左に1 / Right: 右に1
  - キーの状態取得には `key.get_pressed()` を使う
5. 半径: 10, 色: 赤の円で爆弾を作り, ランダムな場所に配置する
  - `surface` の黒い部分を透明にするには「`set_colorkey(黒)`」とする
6. 爆弾を移動させる
  - 横方向速度: `vx = +1` / 縦方向速度: `vy = +1`

※ `v` は `velocity` (速度) の意味

# 練習問題: ex04/dodge\_bomb.py

7. こうかんと爆弾が画面の外に出ないようにする
  - こうかんと: 更新後の座標が画面外の場合→更新前に戻す
  - 爆弾: 更新後の座標が画面外の場合→速度の符号を反転する
8. こうかんと爆弾がぶつかったか判定する
  - 判定にはRectクラスのcollidirect()を使用する
  - 接触したら, ゲームオーバーでmain関数からreturnする

※コミットコメントを「3限基本機能実装完了」としよう

そしてリモートリポジトリにプッシュしよう

# 4限：演習課題

# dodge\_bomb.pyを改良する

- 爆弾ゲームに独自の機能を追加し、楽しいゲームを作ろう
- 追加機能の例：
  - 爆弾を複数個にする
  - 時間とともに爆弾が加速するor大きくなる
  - 着弾するところかとい画像が切り替わる

# コーディング時に意識してみよう

- 読みやすさ: 空行・空白, 文中改行の入れ方
- コードの簡潔さ $\longleftrightarrow$ 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
- ループの作り方: `for i in range` / `for x in list`
- ネストの深さ
- コメントの有無
- 修正容易性, 拡張容易性



# ex04/README.mdも忘れずに書く

- 本日の演習で追加実装した機能についての説明を追記してみよう  
例:

```
# 第4回
## 逃げろこうかとん(ex04/dodge_bomb.py)
### ゲーム概要
- ex04/dodge_bomb.pyを実行すると, 1600x900のスクリーンに草原が描画され, こうかとんを移動させ飛び回る爆弾から逃げるゲーム
- こうかとんが爆弾と接触するとゲームオーバーで終了する
### 操作方法
- 矢印キーでこうかとんを上下左右に移動する
### 追加機能
- 爆弾を複数個にする
- 時間とともに爆弾が加速するor大きくなる
### ToDo(実装しようと思ったけど時間がなかった)
- [ ] 着弾するとこうかとん画像が切り替わる
### メモ
```

# 別ブランチで作業しよう

3限に実装したコードを壊さないように、新たなブランチを作成し、新ブランチで作業する

- ブランチを切る: 「`git branch ブランチ名`」
- ブランチを切り替える: 「`git switch ブランチ名`」

例: 「`git branch add_func`」+「`git switch add_func`」

- 別ブランチでも随時add/commitすること
- 追加機能実装が完了したら、  
コミットコメントを「4限追加機能実装完了」としてcommitすること
- そして、「`git switch main`」でmainに戻る

過去のadd\_funcブランチがある場合は、  
削除してから作成する  
or  
別のブランチ名にする

# ブランチをマージしよう

別ブランチでの開発がうまくいったら, mainブランチに戻り,  
別ブランチでの変更履歴をmainブランチに取り込む

- ブランチをマージする: 「`git merge 別ブランチ名`」
- 例: 「`git switch main`」+「`git merge add_func`」

マージが完了したら, 別ブランチは不要なので削除する

- ブランチを削除する: 「`git branch -d 別ブランチ名`」
- 例: 「`git branch -d add_func`」

そしてリモートリポジトリにプッシュしよう

# 5限：グループワーク

追加機能を実装後, mainブランチにmergeしたら, pushする

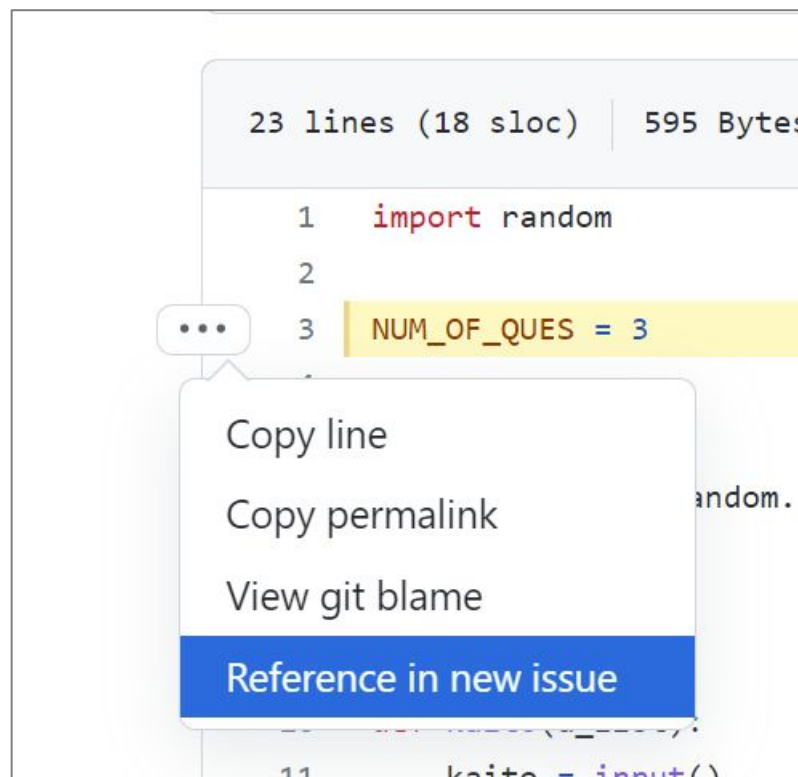
# グループを作り、コードを読む

- Githubで共有されたREADME.mdとコードを読む
- ZOOMのBORでの画面共有により、追加機能をデモする
- コードについて議論する
- 説明者以外の4人のうち最低2人は、コメントをつける
  - ※必ず、全員が2人以上からコメントを受けること
  - ※必ず、全員が2人以上に コメントを付けること
- Issuesでコメントを受けて、
  - 修正すべきなら修正する
  - 修正すべきでないなら、修正しない理由を返信する
- コード修正のコメントがない場合は、TASAからコメントを受けるように、手を上げてお願いする。

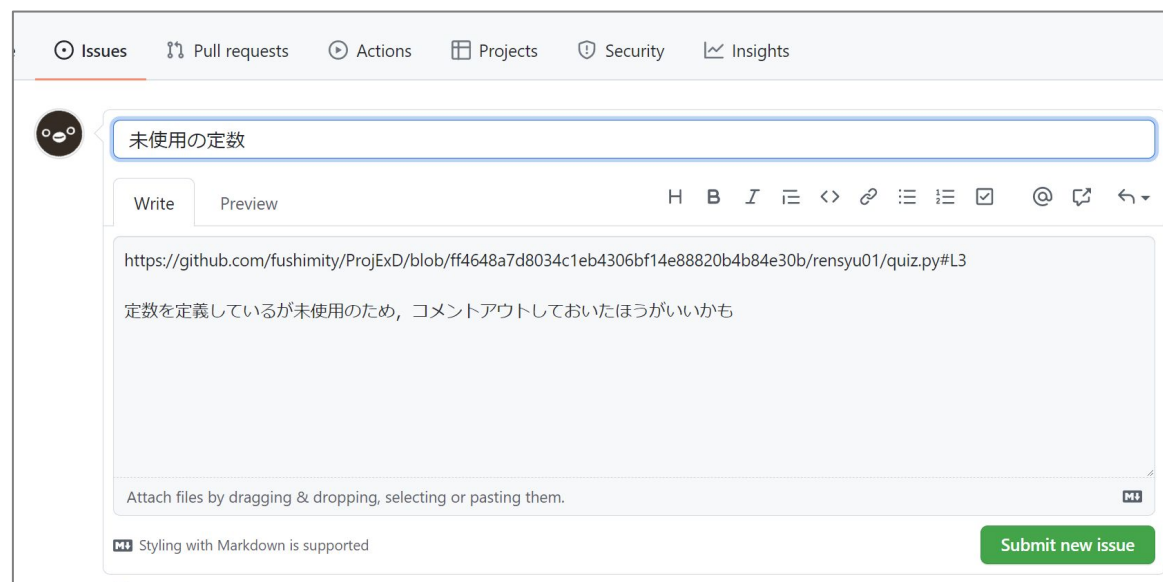
# 【再掲】Issueでコメントを書いてみる

- 当該コードを読み，必要に応じてコメントを送る

①コメントしたい行にカーソルを当て，「...」→「Reference in new issue」

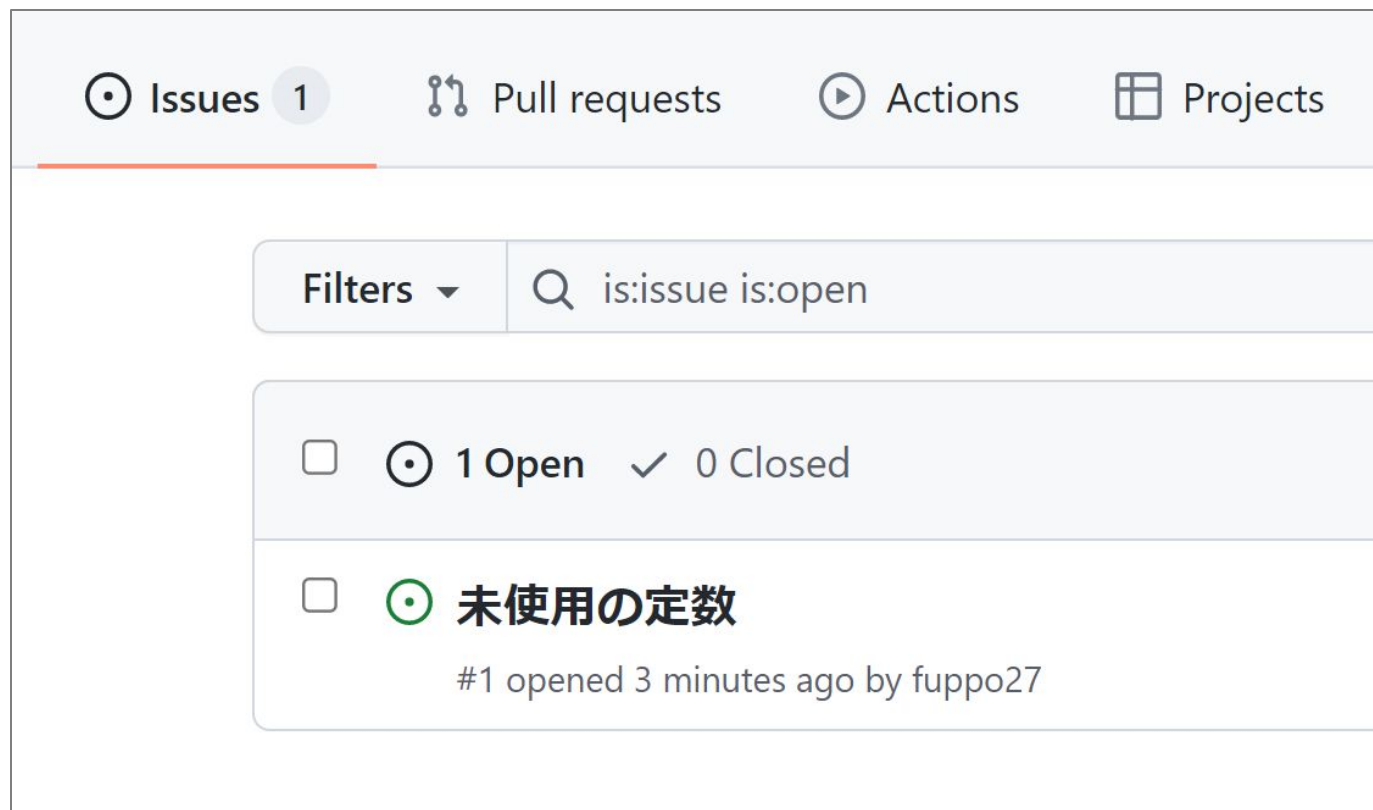


②コメントを書き，「Submit new issue」



# 【再掲】Issueが届く

- Issuesタブから, Issueのコメントを読み, 対応する



# Issueコメントを受けてコードを修正する

1. Issue#1のコメントを受けてコードを修正する
2. 修正が終わったらステージングする「`git add ex04/dodge_bomb.py`」
3. 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, Issue番号を付けること

「`git commit -m "コメント #1 に対する修正"`」

要半角スペース

これにより,  
Issueコメントと対応コミットがGithub上で紐づけられる

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



# 提出物

学籍番号は, 半角・大文字で

- ファイル名: C0B21XXX\_kadai04.pdf

- 内容:

自分のアカウント名

- READMEの最終版(rendered blob)

- <https://github.com/fushimity/ProjExD/blob/main/ex04/README.md>

- コードの最終版

- [https://github.com/fushimity/ProjExD/blob/main/ex04/dodge\\_bomb.py](https://github.com/fushimity/ProjExD/blob/main/ex04/dodge_bomb.py)

- Issues一覧

- <https://github.com/fushimity/ProjExD/issues>

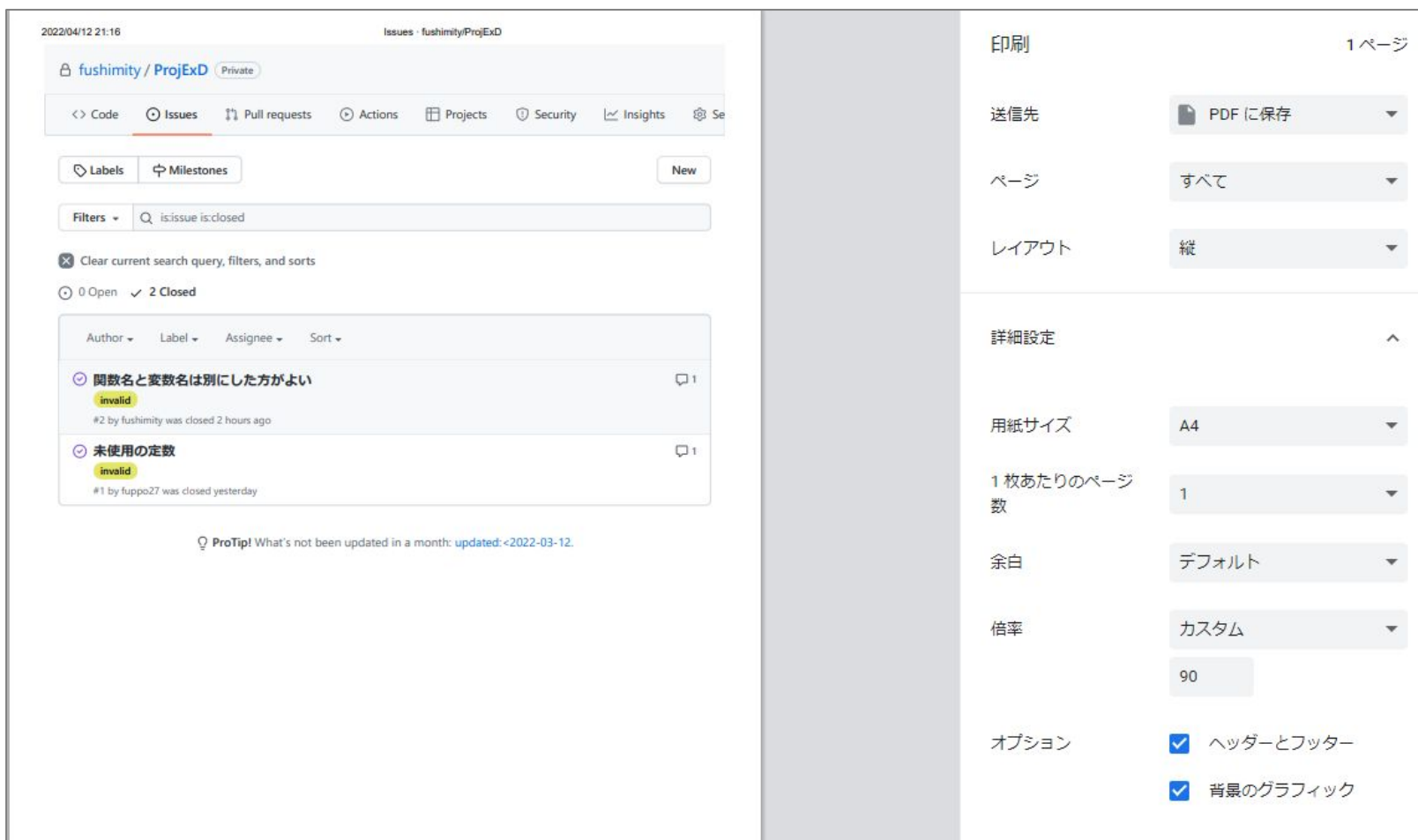
時間内 : 100%  
当日中 : 80%  
今週中 : 50%  
次回まで: 10%

# チェック項目

- READMEの説明は十分か [0 -- 2]
  - わからない [0], まあわかる [1], よくわかる [2]
- 複数の機能を追加しているか [0 -- 4]
  - TASA判断による追加機能の数に基づく. ただし4機能以上は [4]
- Issueで複数人のコメントを受けているか [0 -- 2]
  - コメント人数(TASAを除く)に基づく. ただし2人以上は [2]
- コメントに対して, 適切に反映, 返信しているか [0 -- 2]
  - 対応コメント数に基づく. ただし2コメント以上は [2]
- dodge\_bomb.pyはP.24の点を考慮しているか [0 -- 5]

# 【再掲】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



送信先に「Microsoft Print to PDF」は選ばない(PDFからURLの埋め込みが消える)

←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

# 【再掲】各PDFを単一ファイルにする方法

1. ChromeでPDFとして保存する
  2. 以下のURLから各PDFをマージする
  3. ファイル名を「C0B21XXX\_kadai04.pdf」として保存する
- オンラインでPDFをマージ: [https://www.ilovepdf.com/ja/merge\\_pdf](https://www.ilovepdf.com/ja/merge_pdf)

# 【再掲】提出，チェックの流れ

1. 受講生：提出物ができたらMoodleにアップロードする
2. 受講生：課題チェック依頼のスプレッドシートにて，待ちキューが少ないTASAの列に学籍番号を入力する
3. TASA：Moodleにアップされた課題をチェックする
4. TASA：チェックが済んだら，
  - Moodleに点数を入力する
  - スプレッドシートの学籍番号を赤字に変更する【チェック完了の合図】
5. 受講生：自分の学籍番号が赤字になったら帰る
  - チェックを待たず帰ることも可能だが，提出物に不備があっても修正できない