2022年10月18日(火)
3,4,5限 ＠研A302

プロジェクト演習 テーマD
第3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. リアルタイム処理
2. キー入力
3. Tkinterを使ってGUIゲームの開発

3限：tkinterでGUIゲーム

mainブランチになっていることを確認する

ProjExD2022/ex03/フォルダを作成する

リアルタイム処理

- リアルタイム処理：時間とともに処理が進む
  - 例１：ユーザが何もしなくても敵キャラが画面上を動き回る
  - 例２：背景の雲が流れる
  - 例３：水面が揺らぐ
- Tkクラスのインスタンスメソッド：after(*遅延時間*, *関数*)
  - 特定の処理を指定した時間分遅らせて実行する
  - 特定の処理をある時間ごとに定期的に実行する

```
 9  if __name__ == "__main__":
10      root = tk.Tk()
11      label = tk.Label(root, font=("", 80))
12      label.pack()
13
14      tmr = 0
15      root.after(1000, count_up)
16      root.mainloop()
```

```
3  def count_up():
4      global tmr
5      tmr = tmr+1
6      label["text"] = tmr
7      root.after(1000, count_up)
```

← 1000ミリ秒後にcount_up関数を呼び出す↑

グローバル変数とローカル変数

- グローバル変数：関数の外で定義された変数，どの関数からでもアクセスできる
- ローカル変数：関数の中で定義された変数で，その関数の中からのみアクセスできる

```
 9  if __name__ == "__main__":
10      root = tk.Tk()
11      label = tk.Label(root, font=("", 80))
12      label.pack()
13
14      tmr = 0
15      root.after(1000, count_up)
16      root.mainloop()
```

← 関数の外で定義されている

tmrはグローバル変数なので，
どの関数（count_up関数）からでも
自由にアクセスできる

```
3   def count_up():
4       global tmr
5       tmr = tmr+1
6       label["text"] = tmr
7       root.after(1000, count_up)
```

グローバル変数の値を読み取るだけならglobal宣言不要
グローバル変数の値を書き換えるならglobal宣言必要

global宣言せずに値を更新しようとすると

- 5行目でtmrに値を代入しているので，tmrがローカル変数として解釈される
- そうすると，右辺のtmr（ローカル変数）に値が設定（assignment）されていないことになる
- 値が設定されてないので右辺の計算ができなく，当然代入もできない

```
3  def count_up():
4      #global tmr
5      tmr = tmr+1
6      label["text"] = tmr
7      root.after(1000, count_up)
```

← tmrをglobalであると宣言すれば，tmrがグローバル変数として解釈される
← 右辺のtmr（グローバル変数）に0が設定（assignment）されているので，計算できる

```
  File "c:\Users\fsmtkys\Desktop\ProjExD2022\rensyu03\hello.py", line 5, in count_up
    tmr = tmr+1
UnboundLocalError: local variable 'tmr' referenced before assignment
```

【再掲】イベントの紐づけ（bind）

- マウスの{左，右}クリック，キーの押下などのユーザによるイベントが発生した時に実行される関数を，イベントと紐づける

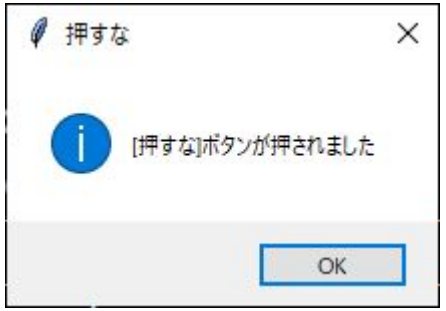
  widget.bind("<イベント>"，イベント発生時に呼び出される関数名)

```
19    button = tk.Button(root, text="押すな")
20    button.bind("<1>", button_click)
21    button.pack()
```

← "<1>"=左クリックが起きると，button_click関数が呼び出される

```
6   def button_click(event):
7       btn = event.widget
8       txt = btn["text"]
9       tkm.showinfo(txt, f"[{txt}]ボタンが押されました")
```

← 左クリックされたウィジェットを表す
← 対象ウィジェットのtext属性の値



※bindのわかりやすいサイト
:https://office54.net/python/tkinter/tkinter-bind-event

キー入力

- widget.bindメソッドの第1引数
  - <Key>：キーを押したとき
  - <KeyPress>：キーを押したとき
  - <KeyRelease>：キーを離したとき

```
22      #root.after(1000, count_up)
23      root.bind("<KeyPress>", key_down)
```

← 1秒後に自動でcount_upが開始
← キーが押されたらkey_downを呼び出し

- eventオブジェクトの**keysym属性**により押下されたキーがわかる

```
2   import tkinter.messagebox as tkm
3
4   def key_down(event):
5       key = event.keysym
6       tkm.showinfo("キー押下", f"{key}キーが押されました")
7       root.after(1000, count_up)
```

← key_downの中からcount_upを呼び出し

リアルタイム処理の中断

- **after()メソッド**は，戻り値としてjob_idを返す
- **after_cancel()メソッド**は，job_idを指定することで，job_idが指し示す処理を終了できる

```
4  def key_down(event):
5      global jid
6      if jid != None:
7          root.after_cancel(jid)
8          jid = None
9          return
10     #key = event.keysym
11     #tkm.showinfo("キー押下", f"{key}キ
12     jid = root.after(1000, count_up)
```

- jidがNoneでなかったら＝すでにafterにより処理が実行されていたら＝キー押下済みだったら，after_cancel()でキャンセルする

```
14 def count_up():
15     global tmr, jid
16     tmr = tmr+1
17     label["text"] = tmr
18     jid = root.after(1000, count_up)
```

```
26     tmr = 0
27     jid = None
```

- グローバル変数として定義（Noneで初期化）
- 使用する関数の中でglobal宣言する

- job_idを戻り値として返す

Canvasクラス

- コンストラクタに指定する引数
  - 親ウィンドウ
  - width：キャンバスの幅
  - height：キャンバスの高さ
  - bg：背景色

- インスタンスメソッド
  - create_image(*x座標*，*y座標*，image=*PhotoImageクラスのインスタンス*，tag="*画像を表すタグ名*")
  - coords("*タグ名*"，*x座標*，*y座標*)

  ※タグ名は，canvasに描画する図形や画像に付与し，動かしたり消したりするときに使用する

PhotoImageクラス

- コンストラクタに指定する引数
  - file：画像ファイルのパス
- 「fig/5.png」をCanvasに表示させる例

```
tori = tk.PhotoImage(file="fig/5.png")
cx, cy = 300, 400
canvas.create_image(cx, cy, image=tori, tag="tori")
```

←相対パス、絶対パスどちらでも指定可

- 画像ファイルの名前、パスを間違えていないかよく確認する
- PhotoImageインスタンスを作るときの注意点
  - ローカル変数にインスタンスを格納しない
    - ガベージコレクション(GC)により，関数が終わった時点でローカル変数が消されてしまうため，root.mainloop()で描画する予定の画像も消されてしまう(何も表示されない)
    - グローバル変数にインスタンスを格納するとGCで消されない
  - 参考サイト：https://daeudaeu.com/create_image_problem/

1問ずつ，ステージング，コミットしよう

練習問題：ex03/maze.py

・迷路ゲームを実装する

1. ゲーム「迷えるこうかとん」用のウィンドウを生成する

2. 幅：1500，高さ：900，背景色：blackのCanvasを生成する
   - ゲームウィンドウがディスプレイからはみ出す場合は，Windowsの設定からシステム→ディスプレイを開き，拡大縮小とレイアウトから，サイズを100%にする

3. figフォルダ内の好きなこうかとんのインスタンスを生成し，Canvasにおける横：300，縦：400の座標に表示させる
   - 横座標の変数をcx，縦座標の変数をcyとすること

※グローバル変数cx，cyは，
こうかとんの現在地を表す変数である

4. 変数keyを空文字""で初期化する

※グローバル変数keyは，
現在押されているキーを表す変数である

1問ずつ，ステージング，コミットしよう

練習問題：ex03/maze.py

5. 関数key_down()を定義し，"<KeyPress>"イベントと紐づける
   - グローバル変数keyに押されたキーのシンボル**keysym**を代入する

6. 関数key_up()を定義し，"<KeyRelease>"イベントと紐づける
   - グローバル変数keyに空文字""を代入する（どのキーも押されていないことを意味する）

7. 常時起動するリアルタイム処理関数main_proc()を定義する
   - グローバル変数cx,cyの値を適切に増減することでこうかとんを動かす
   - グローバル変数keyの値によって動きを分ける
   - Up：上に20／Down：下に20／Left：左に20／Right：右に20
   - Canvasクラスのインスタンスメソッドcoordsで座標を更新する
   - main_proc()を呼び出し，常時起動するようにする

1問ずつ，ステージング，コミットしよう

練習問題：ex03/maze.py

8. DLしたmaze_maker.pyをex03に配置し，
   maze_makerモジュールをimportする

9. maze_makerモジュールのmake_maze関数を呼び出し，迷路を作る
   - 第1引数：横方向のマス数（15）
   - 第2引数：縦方向のマス数（9）
   - 戻り値：15x9の二次元リストで，要素の値は1：壁／0：床を表す

10. maze_makerモジュールのshow_maze関数を呼び出し，迷路を描画する
    - 第1引数：描画先のcanvasインスタンス
    - 第2引数：壁or床を表す二次元リスト

1問ずつ，ステージング，コミットしよう

練習問題：ex03/maze.py

11．　まずは，移動の仕方を変更する

- 現在：cx,cyを20ピクセルずつ上下左右に移動
- 今後：mx,myを1マスずつ上下左右に移動
  （mx,myとマスの大きさ100からcx,cyを計算する）

← main_proc()関数

- mx,my=1,1で初期化し，cx,cyを計算するように変更する

← __main__

12．　1マス移動する際に，移動先が壁なら移動させない

- 壁or床を表す二次元リストにおいて移動予定のマスの値が床：0だったら移動

← main_proc()関数

※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

maze.pyを改良する

- 迷路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：
  - こうかとんの画像を変更する
  - スタートとゴールを追加する
  - 終了（ゴールにたどり着いたか）判定する　　などなど

【再掲】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←→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

ex03/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3回
## 迷路ゲーム:迷えるこうかとん(ex03/maze.py)
### ゲーム概要
- ex03/maze.pyを実行すると，1500x900のcanvasに迷路が描画され，迷路に沿ってこうかとんを移動させるゲーム
- 実行するたびに迷路の構造は変化する
### 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
- スタート地点とゴール地点の追加:スタート地点をランダムに決定し，ゴール地点をスタートから最も遠い場所に設定する機能を追加した.
### ToDo(実装しようと思ったけど時間がなかった)
- [ ] 自動で動くお邪魔キャラの追加
### メモ
```

【再掲】マークダウン記法

書き方	範例
# 見出し	見出し
- 順序なしリスト	• 順序なしリスト
1. 順序付きリスト	1. 順序付きリスト
- [ ] チェックリスト	☐ チェックリスト
> 引用	引用
太字	太字
斜体	斜体
~~取消線~~	~~取消線~~

[link text](https:// "title")	リンク
![image alt](https:// "title")	画像
`コードブロック`	`コードブロック`
``javascript var i = 0; ```	1 var i = 0;
:smile:	😄

参考URL:https://qiita.com/tbpgr/items/989c6badefff69377da7
https://qiita.com/jkjoluvjlajelljicvjzojieoaid/items/01cd7ff819bc2e69b652

【再掲】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

前回の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

【再掲】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

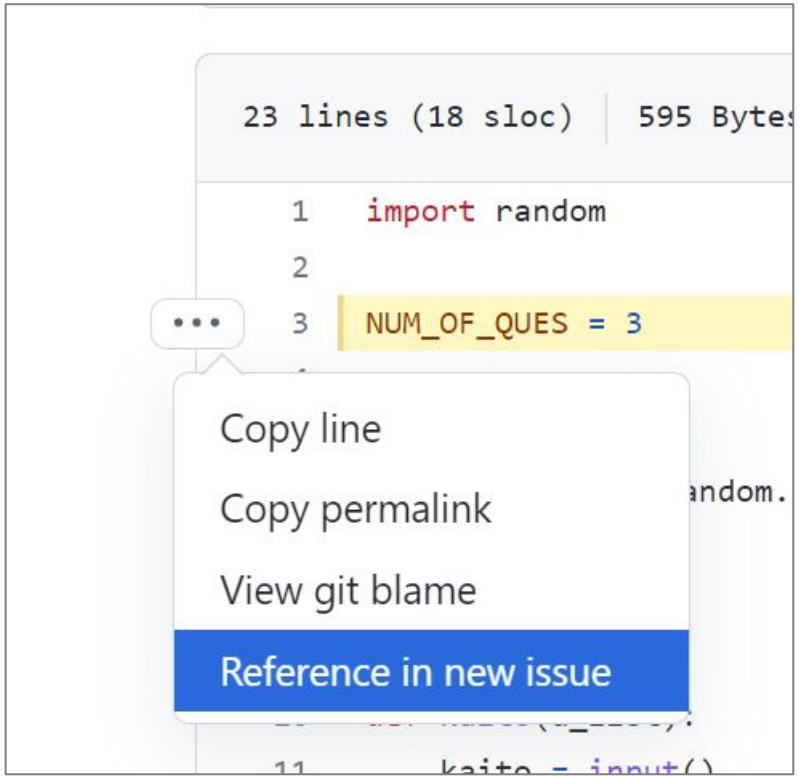
グループを作り，コードを読む

- 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．

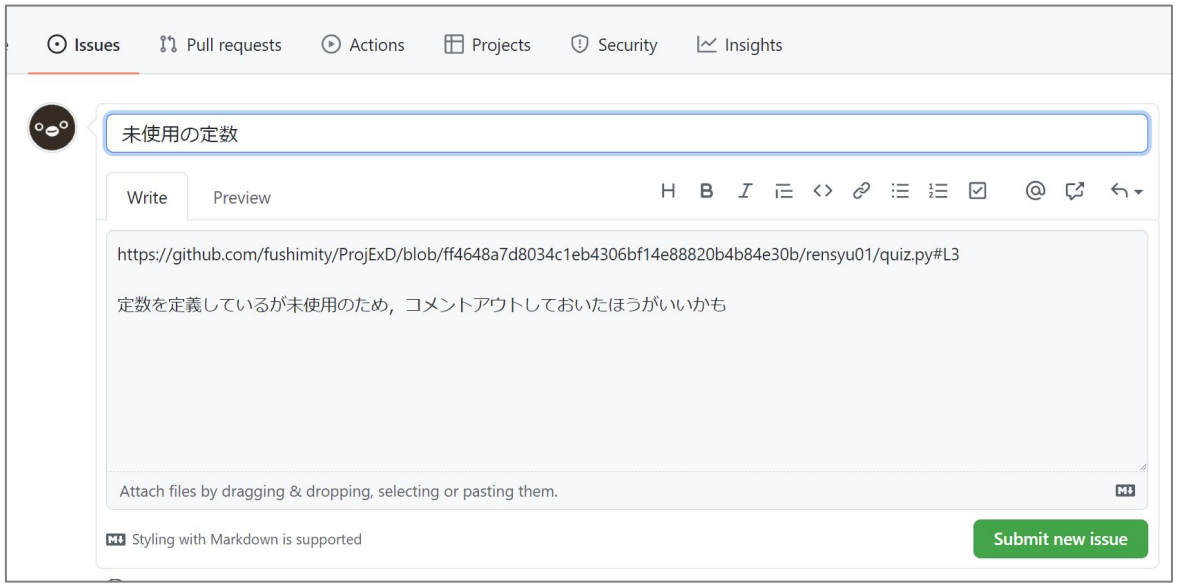
【再掲】Issueでコメントを書いてみる

- 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」

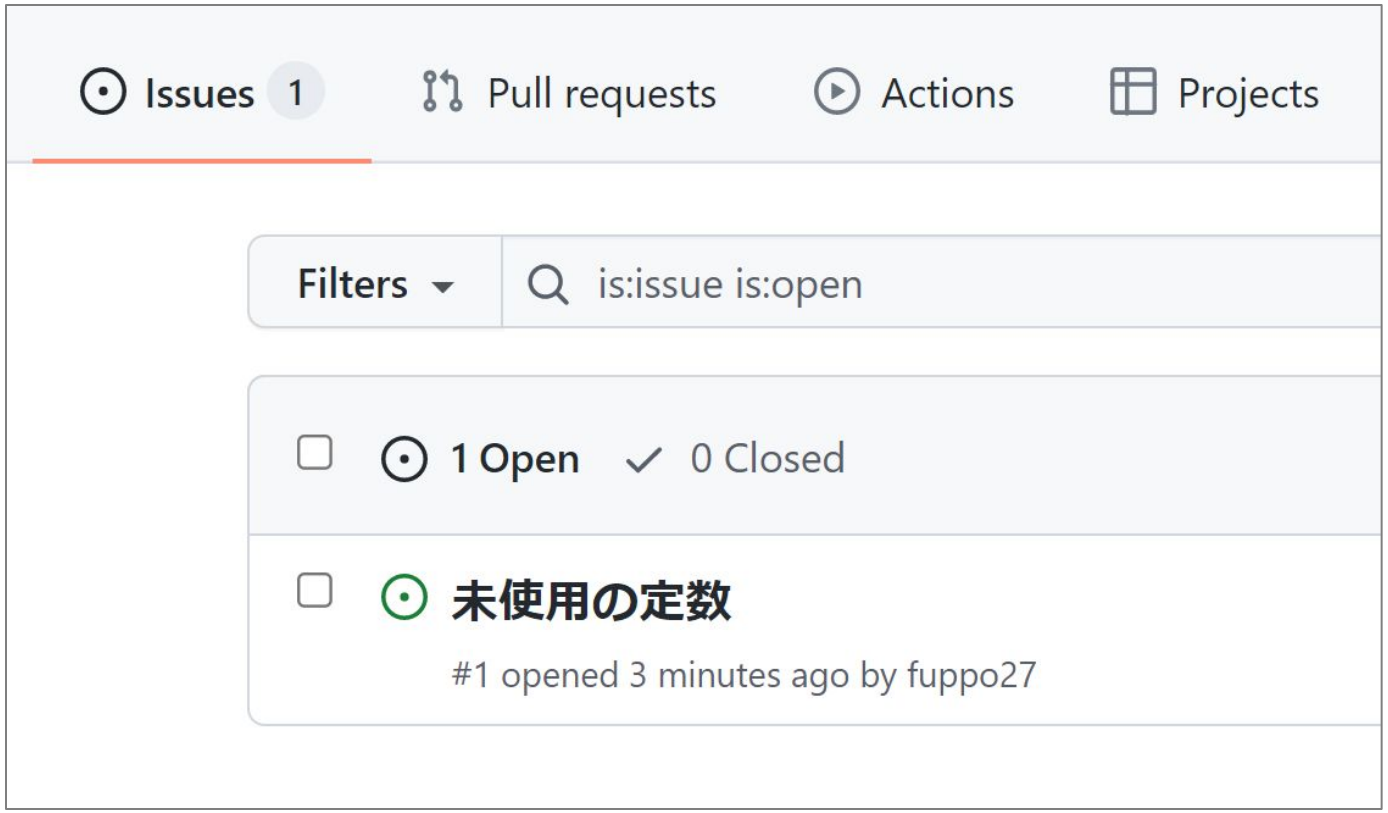


②コメントを書き，「Submit new issue」



【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 ex03/maze.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_kadai03.pdf
- 内容：
  - READMEの最終版（rendered blob）
    https://github.com/fushimity/ProjExD/blob/main/ex03/README.md
  - コードの最終版
    https://github.com/fushimity/ProjExD/blob/main/ex03/maze.py
  - Issues一覧
    https://github.com/fushimity/ProjExD/issues

自分のアカウント名

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], まぁわかる [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]

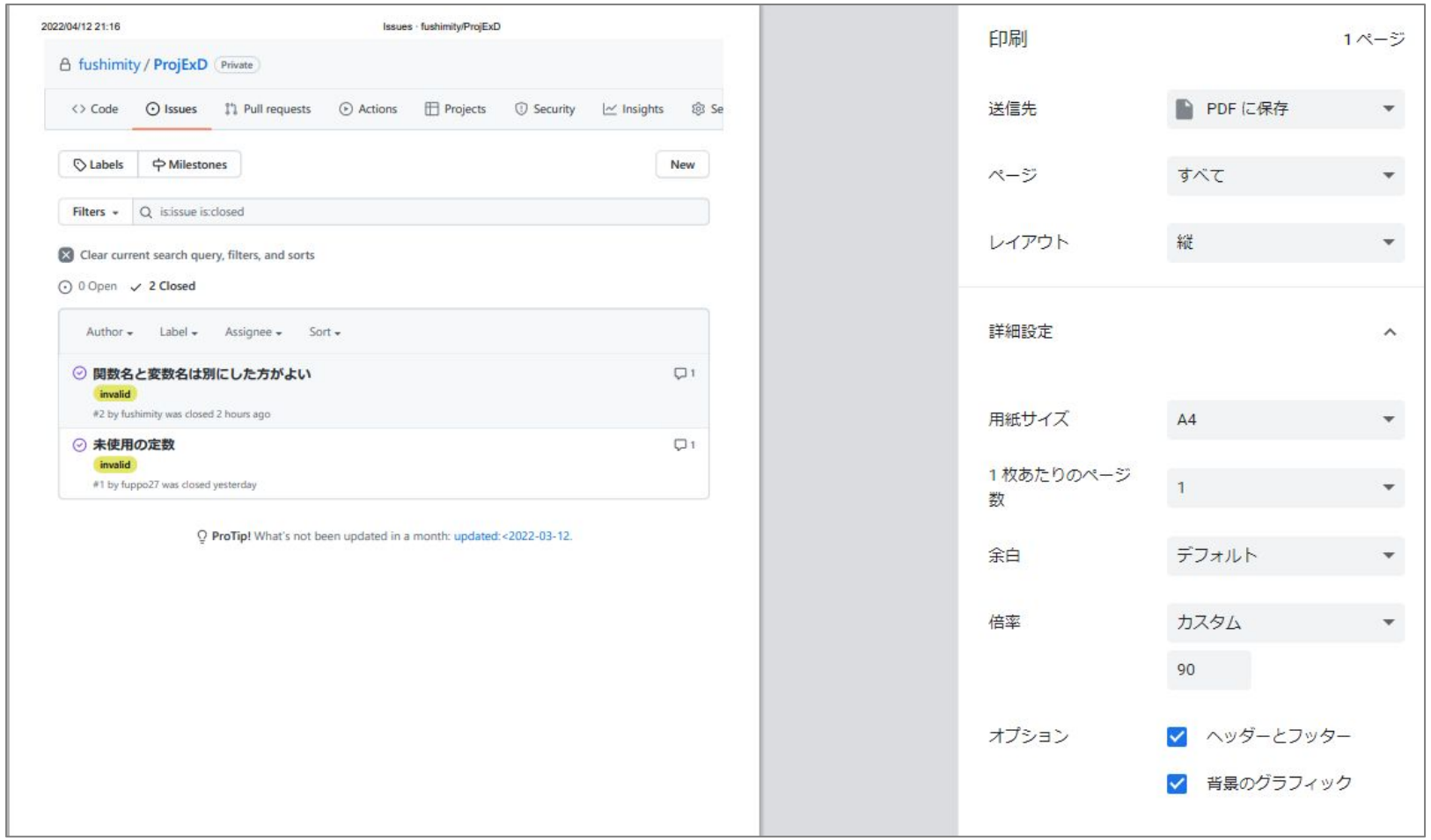
- Issueで複数人のコメントを受けているか [0 -- 2]
  - コメント人数(TASAを除く)に基づく. ただし2人以上は [2]

- 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]

- maze.pyはP.19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_kadai03.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

【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