Diskrétní simulace za použití knihovny SimPy

Panovský Tomáš

12. listopadu 2025

1 Úvod

V této bakalářské práci se zabývám diskrétní simulací za použití knihovny SimPy v Pythonu. Knihovna Simpy umožňuje modelovat procesy, jenž probíhají souběžně, a mohou být zastaveny, nebo pozastaveny na určitou dobu. Praktická část bakalářské práce obsahuje aplikaci simulující průběh hudebního festivalu. Cílem simulace je zjistit, jak se návštěvníci pohybují a kde vznikají fronty, což může pomoci při organizaci reálného festivalu.

2 Systém

str17 Reálnou skutečnost, kterou chceme analyzovat, označujeme jako systém. Analyzovat znamená sledovat, jak se různé části systému chovají v čase, a vyvozovat závěry o efektivitě, kapacitě nebo chování systému.

Na systémy můžeme nahlížet jako diskrétní nebo spojité, přičemž záleží na úhlu pohledu a vlastnostech, které v systému převažují. Diskrétní systém je takový, u kterého se stavové proměnné mění pouze v diskrétních časových okamžicích. Příkladem diskrétního systému je hudební festival, kde se stavové proměnné, například počet návštěvníků ve frontě u stánku s pivem, mění pouze tehdy, když návštěvník přijde na řadu, nebo dostane pivo a odejde. Oproti tomu spojitý systém je takový systém, u kterého se stavové proměnné mění plynule v čase, nikoli pouze v diskrétních okamžicích.

Stav systému je definován jako soubor stavových proměnných, které jsou nezbytné k popisu systému v libovolném okamžiku. V simulaci hudebního festivalu mohou být možné stavové proměnné například počet lidí čekajících ve frontách u stánků s občerstvením, počet lidí právě sledujících koncert, nebo čas příchodu dalšího návštěvníka do areálu.

2.1 Komponenty systému

Každý systém lze chápat jako soubor vzájemně propojených prvků, které společně ovlivňují jeho chování. Aby bylo možné systém lépe pochopit, je vhodné rozdělit jej na základní komponenty, které popisují jeho strukturu a dynamiku. Mezi tyto komponenty patří entity, jejich atributy, aktivity a události.

Entita je objekt zájmu v systému. V našem případě mohou být entitami například návštěvníci festivalu, stánky s občerstvením nebo pódium.

Atribut je vlastnost entity, například počet lidí u stánku, nebo informace o tom, zda má návštěvník hlad nebo je unavený.

Aktivita představuje časově omezenou činnost entity, například čekání ve frontě, sledování koncertu nebo nákup jídla.

Událost je okamžitá změna stavu systému. Události dělíme na Endogenní a Exogenní. Endogenní probíhají uvnitř systému a jsou způsobeny chováním jeho komponent, např. návštěvník dokončí konzumaci jídla a odchází od stánku. Exogenní probíhají v prostředí systému a ovlivňují ho zvenčí, například náhlý déšť.

3 Úvod do simulace

str6 Simulace je napodobení systému v čase. Cílem je vytvořit umělou historii stavu daného systému, a následně pozorovat tuto uměle vytvořenou historii za účelem vyvození závěrů o provozních vlastnostech systému, tedy o měřitelných charakteristikách a výkonnosti systému, jako je například délka front u stánků, průměrná doba čekání návštěvníků nebo hustota návštěvníků u pódia.

Chování systému vyvíjejícího se v čase se zkoumá pomocí simulačního modelu. V této práci je simulační model vytvořen v knihovně SimPy. Simulační model poté může být použit k prozkoumání široké škály otázek typu „co by se stalo, kdyby“ týkajících se reálného systému. Například můžeme zkoumat, zda je daný počet stánků s občerstvením dostatečný pro obsloužení všech návštěvníků festivalu bez dlouhých front.

Možné změny systému pak lze nejprve nasimulovat, aby bylo možné předpovědět jejich dopad na výkonnost systému. Simulace může být také použita ke studiu systémů ve fázi návrhu, ještě před jejich samotnou realizací.

4 Model systému

str. 13 **Model je definován jako reprezentace systému za účelem jeho studia.** Pro většinu studií je nutné zvažovat pouze ty aspekty systému, které ovlivňují problém, který je předmětem zkoumání.

Modely lze klasifikovat jako statické nebo dynamické, deterministické nebo stochastické a diskrétní nebo spojité. Statický simulační model reprezentuje systém v určitém časovém okamžiku, zatímco dynamické modely reprezentují systémy, jak se mění v čase. Deterministické simulační modely neobsahují žádné náhodné prvky, a to ani ve vstupních datech, ani v průběhu samotné simulace. Při opakovaném spuštění se stejnými vstupy poskytují vždy totožný průběh i výsledek simulace. Stochastické modely naproti tomu obsahují jeden nebo více náhodných prvků, které mohou vstupovat do simulace jak na jejím začátku, tak v jejím průběhu, například při generování časů událostí nebo rozhodování o chování entit. Díky tomu lépe vystihují systémy, jejichž chování je ovlivněno náhodou. Výsledky takových simulací nejsou jednoznačné, ale mají pravděpodobnostní charakter. Diskrétní a spojité modely jsou definovány obdobně jako u systémů.

Pro studium hudebního festivalu použijeme diskrétní, dynamický a stochastický model. Diskrétní model, protože stavové proměnné se mění pouze v konkrétních okamžicích. Dynamický model, protože sleduje vývoj systému v čase během celé doby trvání festivalu. Stochastický model, protože některé vstupy, například časy příchodů návštěvníků, doba čekání u stánku nebo délka sledování koncertu, jsou náhodné.

5 Simulace v SimPy

Pro realizaci diskrétní simulace jsem zvolil jazyk Python a knihovnu SimPy. Ta je postavena především na generátorech (využívající příkaz `yield`), které umožňují popis aktivit a událostí entit.

5.1 Generátory v Pythonu

V jazyce Python je iterátor objekt, který umožňuje postupné získávání hodnot bez nutnosti mít všechny hodnoty uložené v paměti. Iterátor si pamatuje svůj aktuální stav a při každém volání funkce next() vrací další prvek.

Zvláštní formou iterátorů jsou generátory, které obsahují v těle funkce příkaz `yield`. Příkaz `yield` umožňuje generátoru postupně produkovat jednotlivé prvky

posloupnosti a může teoreticky produkovat i nekonečnou posloupnost dat. Posloupnost zde znamená řadu hodnot, které generátor postupně poskytuje. Příkaz produkující prvek posloupnosti nabývá tvaru: `yield element`. Po provedení příkazu yield se generátor pozastaví a vrátí hodnotu specifikovanou příkazem `yield`. Při dalším volání pokračuje ve vykonávání od místa, kde byl přerušen.

Příklad generátoru:

```
def get_numbers():
   i = 0
   while True:
      yield i
      i = i + 1
```

Napřed pouze vytvoříme iterátor i:

```
i = get_numbers()
```

Další prvek můžeme vyžádat funkcí next. Dalším prvkem posloupnosti bude hodnota určená příkazem yield. Tedy:

```
next(i)
```

V tuto chvíli bude v proměnné i uložena 0. Tělo generátoru se začne vykonávat od pozastaveného místa až po příkaz yield. Vykonávání těla generátoru je pozastaveno na řádku:

```
i = i + 1
```

Popsaným způsobem získáme další hodnoty z generátoru, tedy při dalším volání `next(i)` bude v proměnné `i` 1, poté 2, a takto můžeme díky podmínce `while True:` pokračovat do nekonečna.

V kontextu diskrétní simulace v knihovně SimPy jsou generátory využity k modelování aktivit, které představují chování jednotlivých entit, například návštěvníků festivalu. Každý příkaz `yield` v generátoru odpovídá předání řízení simulátoru a obvykle vrací objekt typu `Event`, který reprezentuje událost, na jejíž dokončení proces čeká. Příkladem může být čekání ve frontě u stánku s jídlem, dokončení přípravy jídla či jeho obdržení zákazníkem. Takto lze simulovat souběžné aktivity více entit a stochastické prvky, například náhodné časy příprav nebo příchodů návštěvníků, což odpovídá reálnému chování systému.

Následující příklad ukazuje dva roboty, kteří se pohybují různými směry. Každý robot je reprezentován generátorem, který svou aktivitu na náhodně dlouhou dobu pozastaví a umožní simulátoru, aby mezitím vykonal aktivitu druhého robota.

```
def robot(env, name, direction):
    while True:
        print(f"{env.now}: {name} jde směrem {direction}")
        yield env.timeout(random.randint(1, 3))
        print(f"{env.now}: {name} se zastavil a rozhlíží")

env = simpy.Environment()
env.process(robot(env, "Robot A", "vpravo"))
env.process(robot(env, "Robot B", "vlevo"))
env.run()
```

Tento jednoduchý příklad demonstruje princip přepínání mezi aktivitami entit `Robot A` a `Robot B` pomocí příkazu `yield`. Každý robot je reprezentován vlastním generátorem využívajícím příkaz `env.timeout`, který aktivitu na zadaný čas pozastaví. Generátor v tomto příkladu po náhodné době (z intervalu od 1 do 3 časových jednotek) čekání (`env.timeout`) předá řízení zpět simulátoru. Simulátor následně aktivuje jinou aktivitu, ktera je připravena pokračovat. Díky tomuto mechanismu SimPy simuluje souběžné chování více entit v tomto případě dvou robotů pohybujících se v různých směrech. Výstup tohoto příkladu by vypadal následovně:

```
0: Robot A jde směrem vpravo
0: Robot B jde směrem vlevo
2: Robot A se zastavil a rozhlíží
2: Robot A jde směrem vpravo
3: Robot B se zastavil a rozhlíží
4: Robot B jde směrem vlevo
```

5.2 Základní principy SimPy

Pokud SimPy rozložíme na základní principy, je to jen asynchronní dispečer událostí. Asynchronní znamená, že jednotlivé události nemusí čekat na dokončení událostí jiných entit, ale mohou být pozastaveny a znovu spuštěny v různých časech simulace, aniž by blokovaly události ostatních entit. Simpy generuje události a plánuje je na konkrétní čas simulace. Události jsou řazeny podle simulačního času, priority a rostoucího ID události.

Simulační čas je aktuální čas v rámci simulace, nikoliv fyzický čas. V SimPy je čas bez konkrétní jednotky, můžeme ho chápat třeba jako minuty, hodiny, sekundy,

záleží podle toho, co simulujeme. Události jsou řazeny podle toho, kdy mají nastat, tedy SimPy vždy spustí událost naplánovanou na aktuální čas.

Každá událost v SimPy má prioritu, což je číslo, podle kterého se rozhoduje, která událost se zpracuje dříve, pokud je více událostí naplánovaných ve stejný simulační čas. Ve výchozím nastavení všechny události mají stejnou prioritu, zle ji ale ručně nastavit, což umožňuje jemně řídit pořadí zpracování, například aby určitá událost vždy předběhla jinou událost ve stejném časovém okamžiku. Avšak ve většině simulací, včetně této, není potřeba ruční nastatování priorit.

Každá událost má interní identifikátor (ID), který SimPy používá pro rozlišení dvou událostí, které mají stejný čas a stejnou prioritu. ID se zvyšuje s každou novou událostí, takže SimPy ví, která událost byla vytvořena dříve a měla by být zpracována první.

Každá událost má také seznam callback funkcí, které se spustí, když je událost vyvolána a zpracována smyčkou událostí. V SimPy se každý proces, který vrací (yield) událost, interně přidává do seznamu callbacků této události. Callbacky jsou funkce, které se spustí, když je událost dokončena. Díky tomu aktivita pokračuje přesně tam, kde byla pozastavena. Tento mechanismus umožňuje přirozené střídání aktivit a souběžné chování více entit, aniž bychom museli ručně řídit, kdo právě běží. Callback tedy reaguje na dokončení události. Aktivity, které čekají na událost, jsou jedním typem callbacků. Můžeme si ale callbacky definovat i explicitně, například pokud chceme, aby se po dokončení Timeout události spustila nějaká další funkce. To si ukážeme v následujícím příkladu:

```
def my_callback(event):
    print(f"Událost dokončena v čase {event.env.now}, hodnota: {event.value}")

env = simpy.Environment()
event = env.timeout(3, value=42)
event.callbacks.append(my_callback)

env.run()
```

Vytvořili jsme Timeout událost na 3 časové jednotky. Přidali jsme callback `my_callback`, který se automaticky spustí, když Timeout skončí. Když SimPy zpracuje událost, zavolá callback a vypíše zprávu `Událost dokončena v čase 3, hodnota: 42`. Tento příklad demonstruje, že callback je funkce spouštěná po dokončení události. V SimPy ale callbacky vznikají automaticky, když aktivita vrací (yield) událost, to je ten hlavní mechanismus, díky kterému simulace běží sama.

Komponenty SimPy jsou Environment (prostředí), události a procesní funkce, které napíše programátor vytvářející simulaci. Procesní funkce, tedy python generátorové funkce, které yieldují instance Event, implementují simulační model a

tedy definují chování entit simulace. Environment ukládá tyto události do svého seznamu událostí a sleduje aktuální čas simulace.

5.3 Environment

Environment je simulační prostředí a spravuje čas simulace, plánování a zpracování událostí a poskytuje také prostředky pro postupné provádění simulace. To jsme již vysvětlili v základních principech SimPy. Environmet také obsluhuje kompletní spuštění simulace. Simulace v SimPy se spouští příkazem `env.run()`. Simulaci zle spustit dokud nezůstane žádná událost, a to v případě, že zavoláme `env.run()` bez argumentů. To znamená, že pokud procesy popisující aktivity entit běží v nekonečné smyčce `While True:`, tak simulace nikdy neskončí. Dalším způsobem, jak můžeme simulaci spustit, je dokud simulace nedosáhne určitého simulačního času. Tímto způsobem spustíme simulaci, pokud do argumentu vložíme (until=time), kde time je hodnota int, na které se má simulační čas zastavit, například `env.run(until=10)` nebo dokud nenastane konkrétní událost.

6 Příklad metody

Pro ilustraci uvádím metodu `go_to_festival_area()`, která simuluje průchod návštěvníka vstupním turniketem do areálu festivalu.

```
def go_to_festival_area(self, entrances):

    yield self.festival.timeout(random.expovariate(1/5))
    entrance_id = occupied.index(min(occupied))
    entrance = entrances[entrance_id]
    occupied[entrance_id] += 1

    with entrance.request() as req:
        queue_start = self.festival.now
        yield req

        queue_waiting_time = self.festival.now - queue_start
        entry_time = random.uniform(1, 3)
        yield self.festival.timeout(entry_time)

    self.state["location"] = resources.Location.FESTIVAL_AREA
    occupied[entrance_id] -= 1
```

Metoda `go_to_festival_area()` simuluje vstup návštěvníka jedním z dostupných vstupů. Metoda přijímá dva argumenty, a to instanci návštěvníka - self, a SimPy resource entrances. SimPy resource jsou objekty, ke kterým návštěvníci v simulaci přistupují, entrances je tedy seznam vstupů, mezi kterými si návštěvník může vybrat.

Každý návštěvník dorazí po náhodném zpoždění, které je generováno pomocí `random.expovariate(1/5)`, což modeluje příchody návštěvníků s průměrem 5 časových jednotek. Metoda využívá k simulování zastavení času pro návštěvníka funkci `yield self.festival.timeout()`, která na počet zadaných časových jednotek danou instanci uspí.

Návštěvník si vybere vstup, u kterého je aktuálně nejméně lidí čekajících ve frontě, což je sledováno pomocí globálního seznamu `occupied`. Následně požádá o přístup ke vstupu pomocí `entrance.request()`. Proměnná `queue_waiting_time` uchovává dobu, po kterou návštěvník čekal ve frontě, než se dostal „na řadu“. Po získání přístupu stráví návštěvník náhodnou dobu kontolou během vstupu, simulovanou pomocí `random.uniform(1, 3)` a `yield`

`self.festival.timeout(entry_time)`. Nakonec se počet lidí u daného vstupu sníží, což znamená, že návštěvník dokončil průchod vstupem, a nastaví se mu atribut „location“ na FESTIVAL_AREA.

Tento přístup umožňuje sledovat délku front a čekací doby u jednotlivých vstupů, což poskytuje užitečné informace pro analýzu průchodnosti a vytíženosti vstupů během festivalu.