




SimSGamE : Scheduling simulator for modern game engines

Mustapha Regragui ¹, Baptiste Coye ^{1,2}, Laercio Lima Pilla ^{1,3},
Raymond Namyst ^{1,4,5}, and Denis Barthou ^{1,5,6}

1 Inria Bordeaux, France 2 Ubisoft Bordeaux, France 3 CNRS, France 4 Univ. Bordeaux, France 5 Labri, France 6 Bordeaux INP, France

DOI: [10.21105/joss.04592](https://doi.org/10.21105/joss.04592)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Daniel S. Katz](#) 

Reviewers:

- [@azoitl](#)
- [@hwloidl](#)

Submitted: 29 June 2022

Published: 12 August 2022

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

In partnership with



JOSS Special Issue for Euro-Par 2022 Artefacts
[10.1007/978-3-031-12597-3_7](https://doi.org/10.1007/978-3-031-12597-3_7)

Summary

The video game market is valued at over USD\$100 billion ([Mordor Intelligence, n.d.](#)) and impacts computing both at the hardware and software levels. It produces and sells tens of millions of video games and consoles yearly. The variety of devices running games is very large, from personal computers, consoles and smartphones to Cloud gaming servers. In order to develop these games more easily and allow them to be ported to different platforms, developers use a key software component called the game engine.

Game engines are at the heart of the design of modern video games. They handle most of the components of the game, and allow the generation of each frame (image) displayed on the player screen. They aim to keep a high frame rate (30 to 60 frames per seconds) by scheduling the tasks required to complete such computations.

Each task represents a functionality written by a given team at a given moment in the lifetime of the game engine. These tasks have precedence constraints that must be respected to ensure the execution's correctness, which is organised as a directed acyclic graph (DAG), and they have varied execution times. Scheduling such tasks is NP-hard and has not been studied in detail in the context of game engines. A better understanding of this problem could lead to optimizations for the benefit of players and developers.

Our code is focused on modeling a modern game engine and answering the following three questions:

- Can state-of-the-art scheduling strategies improve the performance of the game engine?
- Can changes in the scheduling mechanism reduce the gap between produced schedules and the critical path?
- Can small changes to the task graph lead to performance improvements?

Statement of need

SimSGamE was created in order to study the scheduling of game engines. Given the lack of studies on this problem, our efforts have been dedicated to finding and adapting algorithms and heuristics proposed in other contexts. We find that there is value in bringing to light new applications and knowledge about existing algorithms.

The software simulates the behavior of game engines using two files:

- a dependency file describing links between tasks and the structure of the game engine to be simulated, and
- a tasks file describing key values for tasks (steps, mean, minimum, maximum, std_dev) for both moderated and intensive use of the engine.

Using these two files, SimSGamE generates randomized tasks composing a frame respecting the statistical repartition and their dependencies (generation may be seeded in order to enable direct comparisons). The simulation also mimics different load evolution (from moderate use 0% to intensive 100%) and evaluates multiple scheduling techniques to estimate frame span and metrics, giving information regarding heuristics efficiencies.

The simulation is composed of 200 generated frames with the load parameter starting at 0 (0%) and increasing linearly up to 1 (100%) in the 101st frame and then decreasing linearly until it reaches 0.01 (1%) for the last frame. This provides a gradual change of load while also generating a load peak. In order to have a better understanding of possible optimizations and their impact on the game engine, we also added the ability to sort subtasks (parallel for situations) by non-increasing order of execution time. We consider this is a feasible change to the game engine because it does not affect the actual execution of the subtasks nor the dependencies in the task graph.

The number of available CPUs can be changed to fit the device to emulate or anticipate the effects of external interference but communication and locks are not taken into account at the moment.

The algorithms implemented in the code are:

- FIFO: First In First Out
- LPT: Longest Processing Time First
- SPT: Shortest Processing Time First
- SLPT: LPT at a subtask level (input method name: SLRT)
- SSPT: SPT at a subtask level (input method name: SSRT)
- HRRN: Highest Response Ratio Next
- WT: Longest Waiting Time First
- HLF: Hu's Level First with unitary processing time of each task
- HLFET: HLF with estimated times (input method name: Hu)
- CG: Coffman-Graham's Algorithm (input method name: Coffman)
- DCP: Dynamic Critical Path Priority (input method name: Priority)

In addition, the following algorithms are implemented but not used in automatic tests:

- WL: Weighted Length algorithm
- LFF: Latest Finished First
- MostScussors: Most successors First
- TOPO: topological order
- NEH: Nawaz, Enscore, Ham algorithm
- ACO: Ant Colony Optimization
- Infinity: CPU amount is infinite

Metrics

- SF: slowest frame (maximum frame execution time)
- DF: number of delayed frames (with 16.667 ms as the due date)
- CS: cumulative slowdown (with 16.667 ms as the due date)

State of the field

Scheduling is a well known issue that has been widely studied. However, given the lack of studies on this particular scheduling problem, our efforts have been dedicated to finding and adapting algorithms and heuristics proposed in other contexts. Video games work as soft real-time interactive simulations (Gregory, 2018). However, Real-Time scheduling often considers recurring independent tasks where earliest deadline first (EDF) heuristics are employed (Nascimento & Lima, 2021). Nonetheless, the game engine contains dependent tasks with an entire task graph to be computed at each frame and all tasks share the same due date,

obstructing the use of EDF heuristics. Moreover, parallel task scheduling usually models tasks using multiple resources simultaneously, but the tasks follow a fork-join model internally. An algorithm called DynFed was proposed to schedule parallel tasks with dependencies in real time systems by Dai et al. (2021), but it focuses on periodic, independent tasks whose parallel subtasks have dependencies, while our problem contains periodic tasks with dependencies whose parallel subtasks are independent. The Game Task Scheduler developed by Intel (Alfieri, 2019) reflects these tasks and subtasks organisation but is developed as a tool to be integrated in a game engine and not as a benchmark system that allows testing different heuristics and evaluating performance.

Acknowledgements

We acknowledge contributions from Cédric Dumondelle and Hervé Hubele during the genesis of this project.

References

- Alfieri, B. (2019). *Intel games task scheduler*. <https://github.com/GameTechDev/GTS-GamesTaskScheduler>.
- Dai, G., Mohaqeqi, M., & Yi, W. (2021). Timing-anomaly free dynamic scheduling of periodic DAG tasks with non-preemptive nodes. *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 119–128. <https://doi.org/10.1109/RTCSA52859.2021.00022>
- Gregory, J. (2018). *Game engine architecture* (3rd ed.). Taylor; Francis Ltd.
- Mordor Intelligence. (n.d.). *Global Gaming Market - Growth, Trends, Covid-19 Impact, and Forecasts (2022–2027)*. <https://www.mordorintelligence.com/industry-reports/global-gaming-market>.
- Nascimento, F. M. S., & Lima, G. (2021). Effectively scheduling hard and soft real-time tasks on multiprocessors. *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 210–222. <https://doi.org/10.1109/RTAS52030.2021.00025>